UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TRABAJO FIN DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

SISTEMA ELECTRÓNICO PARA LA REALIZACIÓN FLEXIBLE DE REDES NEURONALES

Memoria y Anexos

Autor: Xinqiu Ye
Director: Jordi Cosp Vilella

Convocatoria: Junio 2018

Resum

El present treball consisteix en el disseny, simulació i test experimental d'un sistema digital integrat en un sol xip que és capaç d'emular el comportament d'una xarxa neuronal, en concret el model anomenat LEGION (Locally Excitatory Globally Inhibitory oscillator Network), dins els múltiples models de xarxa neuronal artificial que s'han publicat en l'actualitat. Aquest model és especialment útil per a resoldre problemes d'enginyeria com la segmentació de patrons o la segregació de figures. Per a la realització de la xarxa es va utilitzar 'Integrate-and-fire' com a model de la neurona. I per al disseny del sistema s'ha utilitzat el llenguatge de descripció d'alt nivell VHDL. Finalment, per comprovar el funcionament del sistema s'ha implementat una xarxa neuronal que reconeix diferents figures i patrons en una imatge d'escala de grisos, sobre un dispositiu lògic programable FPGA. Es pretén que el sistema sigui escalable i eficient alhora, utilitzant el mínim recurs possible.

Resumen

El presente trabajo consiste en el diseño, simulación y test experimental de un sistema digital integrado en un solo chip que es capaz de emular el comportamiento de una red neuronal, en concreto el modelo llamado LEGION (Locally Excitatory Globally Inhibitory Oscillator Network), dentro de los múltiples modelos de red neuronal artificial que se han publicado en la actualidad. Dicho modelo es especialmente útil para resolver problemas de ingeniería como la segmentación de patrones o la segregación de figuras. La red se ha basado en el modelo 'Integrate-and-fire' de la neurona. Se ha utilizado el lenguaje de descripción de alto nivel VHDL para el diseño del sistema. Finalmente, para comprobar el funcionamiento del sistema se ha implementado una red neuronal que reconoce diferentes figuras y patrones en una imagen de escala de grises, sobre un dispositivo lógico programable FPGA. Se pretende que el sistema sea escalable y eficiente a la vez, utilizando el mínimo recurso posible.

Abstract

The present project consists in the design, simulation and experimental test of a digital system integrated in a single chip which can emulate the behavior of a neural network, the model called LEGION (Locally Excitatory Globally Inhibitory Oscillator Network), within the multiple artificial neuronal network models that have been published. This model is especially useful for solving engineering problems such as segmentation of patterns or segregation of figures. For the realization of this network, 'Integrate-and-fire' was used as the model of the neuron. As for the design of the system, the high-level description language VHDL has been used. To check the functioning of the system, a neural network has been implemented, which recognizes different figures and patterns in a gray scale image, on a FPGA programmable logic device. It is intended that the system is scalable and efficient at the same time, using the minimum possible resource.

Agradecimientos

Desarrollar este trabajo ha supuesto un aprendizaje intento y profundo, no solo en el ámbito de electrónica sino también a nivel personal. Por eso me gustaría agradecer a todas aquellas personas que me han ayudado y apoyado durante la elaboración de este.

Primero de todo, querría agradecer profundamente a mi director del TFG, Jordi Cosp Vilella, sin su dedicación, consejo, paciencia y apoyo este trabajo no hubiera salido de forma exitosa.

En segundo lugar, me gustaría agradecer al departamento de Electrónica, por proveer todas las instalaciones y materiales necesarios para desarrollar este trabajo.

También me gustaría agradecer a mi familia y sobre todo a mi hermano Jingfan por su apoyo.

Por último, quiero agradecer a mi mejor amigo Jordi Fornt, por su apoyo y su ayuda con el paquete de software Xilinx Vivado.

Índice

RESUM I

RESUMEN II

ABSTRACT III

AGRADECIMIENTOS V

ÍNDICE VII

1. PREFACIO 9

2. INTRODUCCIÓN 10

2.1. Objetivos del trabajo 10

2.2. Alcance del trabajo 10

3. ESTUDIO BIOLÓGICO DE UNA NEURONA 12

3.1. El sistema nervioso 12

3.2. La neurona 12

3.2.1. Mecanismos de transmisión de la neurona 13

4. REDES NEURONALES ARTIFICIALES 17

4.1. Modelos de neurona 18

4.1.1. Neurona de McCulloch-Pitts 18

4.1.2. Integrate-and-fire 18

4.1.3. Leaky Integrate-and-Fire 19

4.1.4. Hodgkin-Huxley 19

4.2. Modelos de redes neuronales 21

4.2.1. Perceptrón Multicapa (MLP) 22

4.3. Mecanismos de aprendizaje 24

5. DISEÑO Y ARQUITECTURA DE LEGION 26

5.1. LEGION: Principios de funcionamiento 27

5.2. Modelo de la neurona 28

5.2.1. Integrate-and-fire LEGION 29

5.2.2. Adaptaciones del modelo 30

5.2.3. Diseño y arquitectura de la neurona 31

5.3. Red neuronal LEGION 41

5.3.1. Globally Inhibitory Unit 41

5.3.2. Diseño y arquitectura de la LEGION 42
5.3.3. Mecanismos de reset 43
5.3.4. Adaptaciones del diseño para la implementación 45
5.3.5. Simulaciones y resultados experimentales 47

6. IMPLEMENTACIÓN DE LEGION EN FPGA 58

6.1. Entradas 58
6.1.1. Imágenes de entrada 58
6.1.2. Selección de la imagen 59
6.2. Salidas 60
6.2.1. Selección de patrones encontrados 61
6.2.2. Puerto PMOD 63
6.2.3. Display de 7 segmentos 64
6.2.4. Puerto VGA 66
6.3. Arquitectura de todo el sistema 73

7. RECONOCIMIENTO DE PATRONES CON LEGION 76

7.1. Resultados experimentales en el osciloscopio 77
7.2. Resultados experimentales en el monitor 83

CONCLUSIONES 87

PRESUPUESTO 89

BIBLIOGRAFÍA 91

ANEXOS 93

A1. Manual de usuario 93
A2. Código en VHDL de la LEGION 96
A3. Esquema de los bloques que componen al sistema 151

1. Prefacio

Desde el origen de los tiempos, los seres humanos fueron siempre buscando formas de mejorar sus condiciones de vida, desarrollando herramientas para ahorrarse de los trabajos físicos más pesados o peligrosos. Estos progresos han permitido el descubrimiento de otras formas de ahorrarse trabajos no físicos, como cálculos. Como ejemplo de tales progresos, están las máquinas calculadoras, que ayudan a resolver determinados problemas de forma muy rápida y automática. Estas máquinas pueden llegar a manejar más de miles de millones de operaciones por segundo.

Sin embargo, ni el superordenador más potente del mundo es capaz de resolver un problema tan simple para los humanos como es clasificar objetos por sus rasgos comunes, reconocer el gesto de una persona (si está contenta, enfadada o triste) o tener una conversación fluida con el interlocutor. Pues las máquinas solo ejecutan las operaciones con un algoritmo prediseñado, y no son capaces de resolver ningún problema que no sea previsto por el diseñador. Ningún algoritmo lógico puede describir los problemas anteriormente mencionados, ya que hay una cantidad inmensa de variables implicadas y no existe un límite claro entre los diferentes casos, por ejemplo, no todas las personas sonríen de la misma forma y cada uno expresa una misma idea de formas muy diferentes. El motivo de esta incapacidad de las máquinas se esconde en que no tienen una memoria donde recoger la experiencia pasada, ni tienen la capacidad de hacer conclusiones basando en dicha experiencia, algo que los humanos sí lo poseemos.

La inteligencia artificial nace de estas limitaciones, intentando descubrir y describir ciertas características de la inteligencia humana para que las máquinas puedan simularlas. Una de las ramas en la disciplina de inteligencia artificial es redes neuronales artificiales, donde se intenta emular el funcionamiento del cerebro humano e implementarlo en una máquina, con el fin de poder resolver estas tareas que antes no eran posible resolver.

De esta motivación nace el presente proyecto. En los capítulos siguientes, se va a presentar los aspectos fundamentales de una red neuronal biológica y los modelos que simulan esta red con mayor o menor realismo. Y finalmente, una realización física de la red neuronal sobre un sistema electrónico.

2. Introducción

Tal y como se ha introducido en el prefacio, sería muy interesante poder reproducir ciertas características de la inteligencia humana para aplicaciones o problemas que no puedan expresarse con un algoritmo. En este proyecto, se intenta implementar una red neuronal artificial en un dispositivo programable lógico FPGA para aplicaciones de esta característica. Para ello, se definen a continuación los objetivos y el alcance del trabajo.

2.1. Objetivos del trabajo

Para poder diseñar e implementar una red neuronal artificial suficientemente robusta y eficiente, se necesita conseguir los siguientes objetivos:

- Estudiar el funcionamiento biológico de una neurona y sus mecanismos de transmisión de información.
- Buscar información sobre diferentes modelos matemáticos de una neurona
- Coleccionar información sobre diferentes modelos de redes neuronales artificiales
- Seleccionar el modelo más flexible y eficiente para realizar
- Diseñar el sistema digital haciendo adaptaciones necesarias del modelo original
- Describir el sistema mediante el lenguaje VHDL
- Verificar el correcto funcionamiento del sistema mediante simulaciones
- Implementar el sistema en un dispositivo FPGA y comprobar su correcto funcionamiento
- Hacer modificaciones o adaptaciones necesarias para comprobar la escalabilidad del sistema

2.2. Alcance del trabajo

El presente proyecto contempla todo el diseño de la red neuronal artificial, basándose en una estructura modular para conseguir mayor flexibilidad y escalabilidad. Teniendo en cuenta las limitaciones de recursos y tiempo, el proyecto debe cumplir como mínimo lo siguiente:

- Desarrollar una red neuronal de tamaño razonable, de entre 100 y 1000 neuronas. Ya que una red de menos de 100 neuronas no tiene suficiente interés práctico
- La implementación de la red en el sistema electrónico debe tener una interface de usuario para poder controlar parte del sistema y poder visualizar los resultados
- La red debe tener tolerancia a fallos: dar un resultado parcial en caso de producirse un daño en parte del sistema
- La red debe tener tolerancia a interferencias en la entrada

3. Estudio biológico de una neurona

En este apartado se va a explicar las ideas básicas sobre el sistema nervioso y el mecanismo de transmisión de información de su componente más importante, la neurona.

3.1. El sistema nervioso

El sistema nervioso, presente en la mayoría de los animales, está compuesto principalmente por dos tipos especializados de células [1]:

- Las neuronas, que se encargan de recibir estímulos y transmitir señales
- Las células gliales, que sostienen, protegen y alimentan a las neuronas

El sistema nervioso no es igual a una red neuronal, sino que la contiene. En un sistema nervioso completo, éste recibe estímulos de los órganos, sean interiores o exteriores, y son transmitidos por las neuronas sensoriales al sistema nervioso central, que es el cerebro y la médula espinal. De ahí se interpreta la información y mandan los impulsos de respuesta a los órganos de acción mediante neuronas motoras. Si se hace una comparación analógica con un sistema electrónico, los órganos sensoriales serían los sensores, la red neuronal artificial comportaría como el sistema nervioso central y los actuadores, como los órganos efectores.

3.2. La neurona

La neurona es una célula altamente especializada en recibir y transmitir información, mediante señales eléctricas denominadas impulsos nerviosos o potenciales de acción.

La porción más grande de la neurona es el cuerpo de la célula, donde contiene la mayoría de los organelos y el núcleo. Un solo axón se extiende desde el cuerpo de la célula y puede llegar a medir un metro o más de longitud, aunque su diámetro es microscópico, para poder llevar la información a la parte del cuerpo más lejano. El axón se divide en su extremo, formando muchas ramas terminales que acaban en terminales sinápticas, donde se liberan los neurotransmisores. La unión entre una terminal sináptica y otra neurona se denomina sinapsis. Del cuerpo de la célula se prolongan también las dendritas, típicamente cortas y bastante ramificadas, dedicadas principalmente a recibir estímulos. [1]

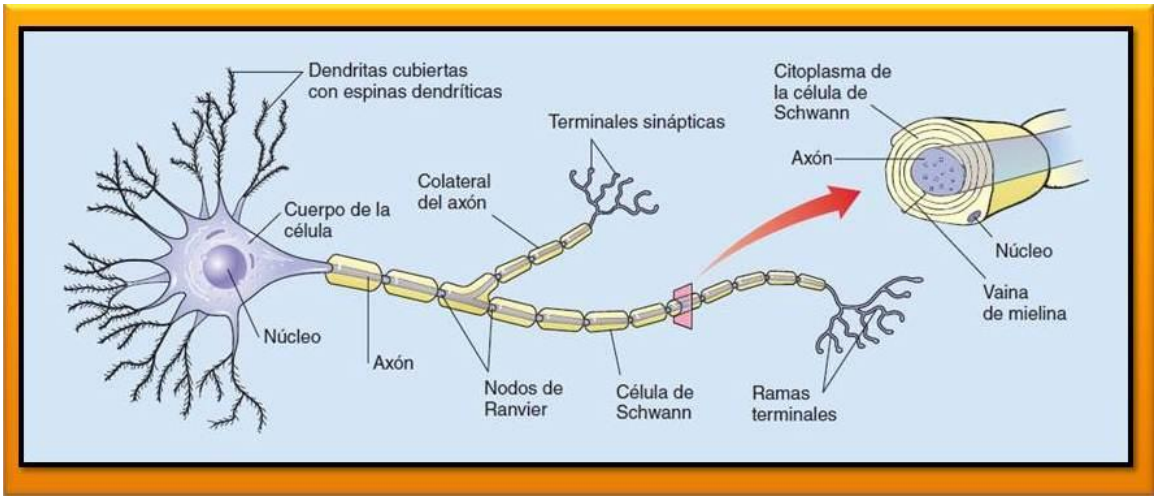


Figura 1 - Partes de una neurona. [2]

3.2.1. Mecanismos de transmisión de la neurona

3.2.1.1. Potencial de reposo de la neurona

Como la mayoría de las células animales, la membrana plasmática de las neuronas está polarizada eléctricamente. Es decir, la carga eléctrica dentro de la célula es más negativa que la carga eléctrica del líquido extracelular. Esta diferencia de carga a través de la membrana se denomina potencial de membrana.

Cuando la neurona no está excitada, ésta tiene un potencial de reposo de aproximadamente 70 mV. Esta diferencia de voltaje es debida a las diferencias en concentraciones de iones dentro y fuera de la célula. Dentro de la célula, la concentración del ion potasio (K+) es alrededor de 10 veces mayor que en su exterior, mientras que la concentración del ion sodio (Na+) es alrededor de 10 veces menor. [1]

Otro factor importante que influye en la magnitud del potencial de membrana es la permeabilidad selectiva de la membrana plasmática hacia estos iones. La membrana plasmática no es de igual permeable para todos los iones. De hecho, es de hasta 100 veces más permeable al ion potasio (K+) que al ion sodio (Na+). De esta forma, el ion Na+ bombeado hacia fuera no puede volver fácilmente hacia la célula, mientras que el ion K bombeado hacia la neurona penetra fácilmente en la célula. [1]

El ion K+ entra en la célula debido al gradiente eléctrico, ya que la tensión dentro de la célula es menor, y difunde hacia fuera de la célula debido al gradiente de concentración, pues hay una menor concentración de iones K+ fuera de la célula, llegando a un punto en que estos dos flujos se igualan, denominado potencial de equilibrio. El potencial de equilibrio del ion K+ es de -80 mV típicamente, y el de ion Na+, -40 mV. El desarrollo del potencial de reposo de la neurona es establecido principalmente por el flujo neto de K+ hacia la neurona, por lo que se aproxima más al potencial de equilibrio del ion

potasio. Una vez alcanzada este potencial de reposo, la situación se mantiene con que las bombas de sodio-potasio en la membrana bombean dos K+ hacia dentro de la célula por cada tres Na+ que bombean hacia fuera. [1]

3.2.1.2. Potencial de acción

Un estímulo puede incrementar el potencial de membrana de la neurona al aumentar la permeabilidad de la membrana plasmática a los iones sodio. En este caso, se dice que la membrana está despolarizada. Ésta puede estar también hiperpolarizada, y disminuye su habilidad para generar un impulso neuronal.

Cuando el estímulo recibido es suficientemente fuerte, la membrana se despolariza llegando al nivel de umbral, donde la neurona dispara un impulso nervioso, o potencial de acción. Esta señal eléctrica se propaga rápidamente por el axón hacia las terminales sinápticas, y por ahí estimula a las neuronas vecinas. El nivel de umbral de casi todas las neuronas está a -55 mV aproximadamente. Si la membrana plasmática de la neurona alcanza a un potencial de tensión mayor a -55 mV con respecto al líquido extracelular, disparará enviando un potencial de acción. En caso de no alcanzar esta tensión, no se generará ningún impulso. Así pues, un potencial de acción es una respuesta total o nula. No existe variación en la intensidad de un solo impulso. Las variaciones de sensaciones no se distinguen por la intensidad del potencial de acción, sino en la cantidad de neuronas estimuladas y en su frecuencia de descarga.

Durante el instante en que la neurona es despolarizada, la membrana del axón está en un período refractario absoluto, sin poder transmitir otro potencial de acción independientemente de la magnitud del nuevo estímulo recibido. Después de cierto tiempo, la membrana se vuelve nuevamente impermeable al ion sodio, iniciando el proceso de repolarización. Para volver a su estado de reposo, la neurona entra en un período refractario relativo, durante el cual la membrana está hiperpolarizada. Puede transmitir impulsos, aunque partiendo de un nivel de potencial más negativo. [1]

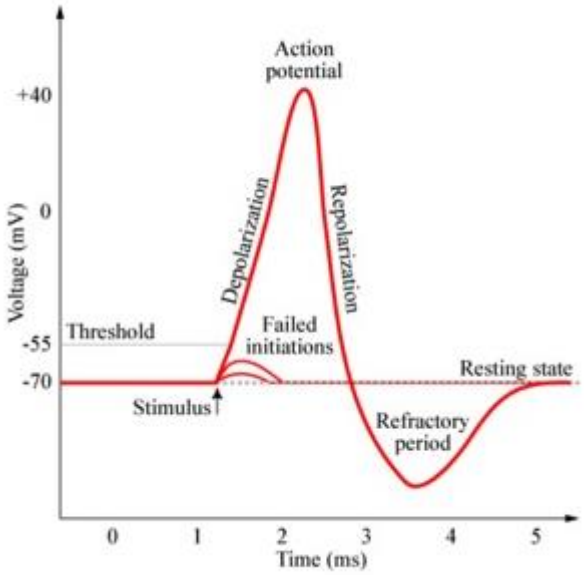


Figura 2 - Estimulación de la neurona: generación del potencial de acción. [3]

En conclusión, las características más importantes de la neurona se pueden resumir en lo siguiente:

- Las dendritas reciben estímulos de neuronas vecinas y las terminales sinápticas transmiten señales a otras neuronas. La unión se denomina sinapsis
- En reposo, la neurona presenta un potencial de -70 mV aproximadamente
- Con estímulos suficientes, el potencial de membrana supera un umbral (-55 mV) a partir del cual se produce un potencial de acción que estimula a las neuronas postsinápticas
- Una neurona solo puede generar un potencial de acción al mismo tiempo. Hay que esperar un período refractario como mínimo para poder iniciar otro disparo
- Tras el disparo, la neurona tarda un tiempo en volver al estado de reposo, pasándose antes a un potencial más negativo

4. Redes neuronales artificiales

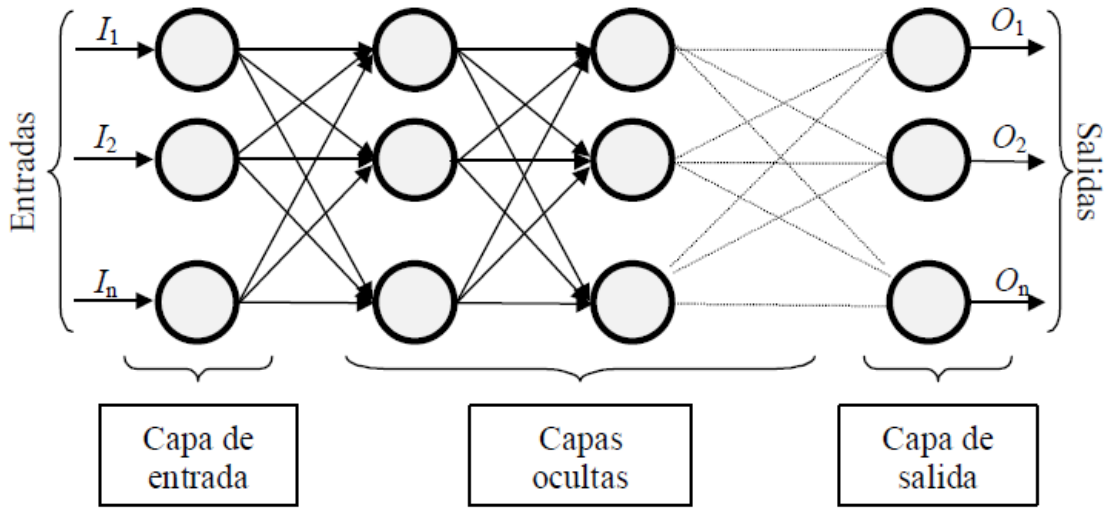


Figura 3- Niveles en una red neuronal. [4]

Una red neuronal puede verse dividida en tres capas o niveles. Las neuronas de la capa de entrada reciben datos de entradas, función similar a las neuronas sensoriales. Estos datos pasan por una o más capas ocultas donde se procesa la información y los resultados son transmitidos hacia el exterior a través de la capa de salida. Haciendo una comparación con el sistema nervioso, las capas ocultas serían el cerebro y la capa de salida, las neuronas motoras.

Dicho esto, para poder implementar una red neuronal, por regla general se necesitan los siguientes elementos:

- Modelo de la neurona, que describe los mecanismos de disparo de las neuronas
- Modelo de red neuronal, que describe la topología con la se organizan las neuronas
- Mecanismos de aprendizaje, con los cuales la red aprende a tomar decisiones propias basándose en el aprendizaje y en la experiencia

4.1. Modelos de neurona

Un modelo de neurona es una descripción matemática que intenta emular el comportamiento de una neurona. A continuación, se presentan con detalle algunos modelos más conocidos.

4.1.1. Neurona de McCulloch-Pitts

El modelo de neurona McCulloch-Pitts fue uno de los primeros modelos de la neurona artificial, introducido por Warren McCulloch y Walter Pitts en 1943.

La neurona de McCulloch-Pitts [5] realiza una suma ponderada de los estímulos recibidos y hace comparar esta suma con un valor de umbral para decidir si se genera un impulso nervioso o no. Aquí se puede observar la característica de respuesta total o nula de la neurona.

$$sum = \sum_{i=1}^{N} w_i \cdot I_i \tag{4.1}$$

$$y = \begin{cases} 0, si\ sum < umbral \\ 1, si\ sum \geq umbral \end{cases} \tag{4.2}$$

4.1.2. Integrate-and-fire

Otro modelo es el Integrate-and-fire, introducido por Louis Lapicque en 1907. En este modelo, a diferencia del de McCulloch-Pitts, la neurona está caracterizada por su potencial de membrana (Vm) que evoluciona con el tiempo según la ecuación diferencial [6]:

$$I(t) = C_m \cdot \frac{dVm(t)}{dt} \tag{4.3}$$

donde Cm es la capacitancia de la membrana celular, Vm es el potencial en la membrana y I(t) es la intensidad de corriente que se aplica a la neurona.

De esta forma, cuando la neurona recibe una intensidad de corriente como estímulo de entrada, su potencial de membrana se incrementa con el tiempo (integración de la corriente recibida sobre el tiempo). Si el potencial supera un umbral se generará un impulso. Después del impulso, el potencial es reseteado a cero y el proceso vuelve a empezar. De ahí viene su nombre, integra y dispara.

El modelo puede ser más preciso si se introduce un período refractario (t_{ref}) que evita el disparo de la neurona durante este tiempo, limitando así su frecuencia de disparo, que se describe como [6]:

$$f(I) = \frac{I}{C_m \cdot V_{th} + t_{ref} \cdot I} \tag{4.4}$$

Donde f es la frecuencia de disparo, V_{th} es la tensión de umbral a partir del cual la neurona pueda disparar.

Cabe destacar de que el modelo Integrate-and-fire no corresponde con el comportamiento real observado de las neuronas en el sentido de que al recibir un estímulo no lo suficiente grande como para producir el disparo, el potencial de la neurona no vuelve al potencial del reposo hasta que se genere un impulso, otorgando a la neurona un potencial más alto del de reposo durante todo el tiempo antes del impuso.

4.1.3. Leaky Integrate-and-Fire

El modelo Leaky Integrate-and-fire [7] es una variante mejorada del modelo Integrate-and-fire. La diferencia está en considerar que la membrana plasmática de la neurona no se comporta como un condensador ideal, sino que presenta cierta resistencia (Rm) que absorbe parte de la corriente recibida, por lo que en este modelo se añade un nuevo sumando, the Leaky Integrator o el integrador de fugas.

$$I(t) = C_m \cdot \frac{dVm(t)}{dt} + \frac{Vm(t)}{Rm} \tag{4.5}$$

Así pues, como parte de la corriente es absorbida por la resistencia de la membrana, la neurona necesita una corriente de entrada un poco más alta para poder generar un impulso.

4.1.4. Hodgkin-Huxley

El modelo de Hodgkin y Huxley (1952) [8] describe los mecanismos iónicos bajo la iniciación y la propagación de potenciales de acción en el axón gigante del calamar, basándose en que las propiedades de un segmento de la membrana neuronal pueden ser modeladas por un circuito eléctrico equivalente como el mostrado a continuación.

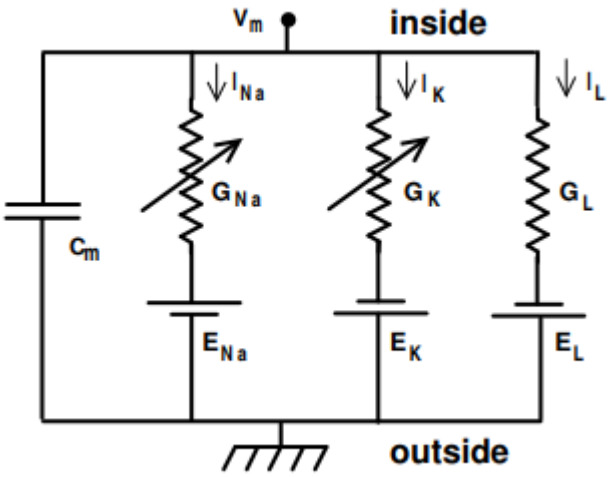


Figura 4- circuito eléctrico equivalente. [9]

En una neurona real, las concentraciones iónicas dentro de la célula son diferente que las del líquido extracelular debido al transporte de iones a través de la membrana plasmática de la neurona. Estas diferencias crean tanto un gradiente de concentraciones como un gradiente eléctrico que hace que el flujo de corriente aplicado se divide en dos componentes, una corriente capacitiva en cargar el condensador (equivalente a la membrana de la neurona) y otro flujo de corriente que se asocia con el movimiento de tipos específicos de iones a través de la membrana. Este comportamiento es descrito con la siguiente ecuación [9]:

$$I_{ext} = C_m \cdot \frac{dVm}{dt} + I_{ion} \tag{4.6}$$

Donde C_m es la capacitancia de la membrana, Vm es el potencial intracelular, I_{ext} es la corriente aplicada externamente y I_{ion} la corriente iónica neta a través de la membrana.

Esta última corriente iónica, es dividida a su vez en tres [9]: corriente asociada con el canal sodio (I_{Na}), corriente asociada con el canal potasio (I_K) y una corriente de fugas (I_L).

$$I_{ion} = \sum_i G_i \cdot (V_m - E_i) \tag{4.7}$$

Donde G es la conductancia del canal i, E es el potencial de equilibrio del canal i, siendo i sodio, potasio o la resistencia de fuga. Desarrollando la ecuación, la ecuación final quedaría:

$$I_{ext} = C_m \cdot \frac{dVm}{dt} + G_{Na} \cdot (V_m - E_{Na}) + G_K \cdot (V_m - E_K) + G_L \cdot (V_m - E_L) \tag{4.8}$$

De esta forma, usando el modelo Hodgkin y Huxley se puede describir con precisión el mecanismo de transmisión en la neurona, aunque a nivel de implementación en hardware es muy costoso debido a que la resolución de la ecuación es muy compleja.

4.2. Modelos de redes neuronales

Se puede decir que el campo de las redes neuronales artificiales se inició con el procesamiento de imágenes en el año 1958, cuando se publicó la primera red neuronal llamada Perceptrón, usado como detector de caracteres ópticos [10]. A partir de entonces se publicaron una gran variedad de modelos de red basándose en diferentes criterios, algunas de ellas son [11]:

- Red Neuronal Celular (CNN)
  En esta red, cada neurona puede interaccionar con sus vecinos. "Contiene una cantidad fija de unidades llamadas células, las cuales son sistemas no lineales y dinámicos donde la información de entrada es codificada para conocer su comportamiento. Cada célula está localmente interconectadas con múltiples entradas que vienen de otras células."[11]

- Redes Neuronales Oscilatorias (ONN)
  Este tipo de red se basa en oscilaciones periódicas sincronizadas de neuronas en forma de grupos en la corteza visual, con un estímulo de entrada similar. De forma que diferentes grupos de oscilaciones indican la presencia de objetos diferentes en una imagen. El modelo más común de ONN es el LEGION, del que se hablará con mayor profundidad en el apartado ?.

- Redes Neuronales de Impulsos (SNN) [12]
  La red neuronal de impulsos, Spiking Neuronal Network (SNN) en inglés, es un tipo de redes neuronales artificiales que más se asemejan a las redes biológicas en comparación con las redes clásicas como Perceptrón, entre otras.
  En este tipo de red, la neurona genera un impulso o potencial de acción con una amplitud y una duración determinadas. Este impulso se transmite a otras neuronas, y éstas pueden aumentar o disminuir su potencial de acuerdo con esta señal. Así pues, las neuronas pueden generar un tren de impulsos que transmite información basándose en la codificación de pulsos: información contenida en la frecuencia y el número de impulsos [13].

- Otras redes
  A parte de las mencionadas anteriormente, existen muchas más redes neuronales basándose en otros criterios. Aquí solo se mencionan algunas de ellas:
    - Red Neuronal Pulso-Acoplada (PCNN)
    - Redes Neuronales Recurrentes (RNN)

- Redes Neuronales basadas en modelos probabilísticos (RPNN)
- Mapa Autoorganizado de Kohonen (SOM)
- Red Neuronal con Funciones de Base Radial (RBFNN)
- Teoría de Resonancia adaptiva (ART)

A continuación, se explica con más detalle el modelo más simple, Perceptrón.

4.2.1. Perceptrón Multicapa (MLP)

El Perceptrón [14] fue la primera red neuronal descrita algorítmicamente, trabajo publicado por Frank Rosenblatt en 1958. Esta red fue construida sobre el modelo de la neurona de McCulloch-Pitts y sirve para clasificar dos clases de elementos.

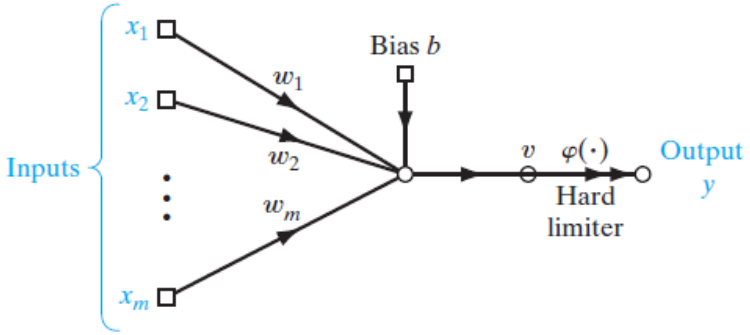


Figura 5 - Perceptrón de una capa. [15]

El Perceptrón consiste básicamente de unas entradas (x_i) con un peso sináptico ajustable (w_i), una señal aplicada externamente (bias), un sumador (v) y un comparador ($\varphi(\cdot)$) en la salida.

$$v = \sum_{i=1}^{m} w_i \cdot x_i + b \tag{4.9}$$

Tras hacer la suma ponderada, el resultado se compara con un umbral. Dependiendo de si está encima o debajo de ese umbral, la salida tomará un valor de los dos posibles, +1 ó -1, representando que la entrada es objeto de clase A o de clase B.

La señal bias de la ecuación es un mecanismo de aprendizaje supervisado para ayudar al Perceptrón a fijar el límite entre estas dos clases [16]. Este límite de decisión se define como [15]:

$$\sum_{i=1}^{m} w_i \cdot x_i + b = 0 \tag{4.10}$$

En la Figura 6 se puede ver un ejemplo de un Perceptrón con dos entradas. Dependiendo del valor de la señal bias, la recta de decisión puede tener una pendiente u otra, afectando la exactitud de la capacidad de clasificación de la red.

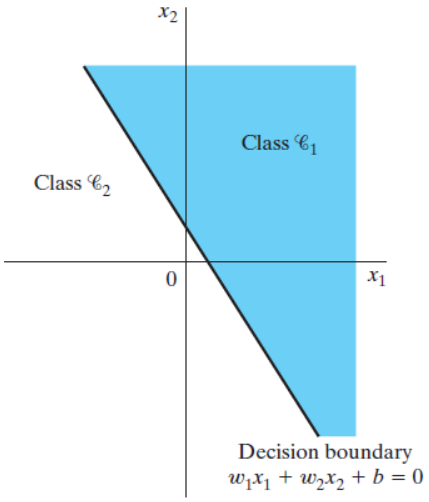


Figura 6 - Límite de decisión de un Perceptrón con dos entradas. [15]

Si se combinan varios perceptrones de una capa, se obtiene un Perceptrón multicapa (MLP), que permite realizar funciones más complejas.

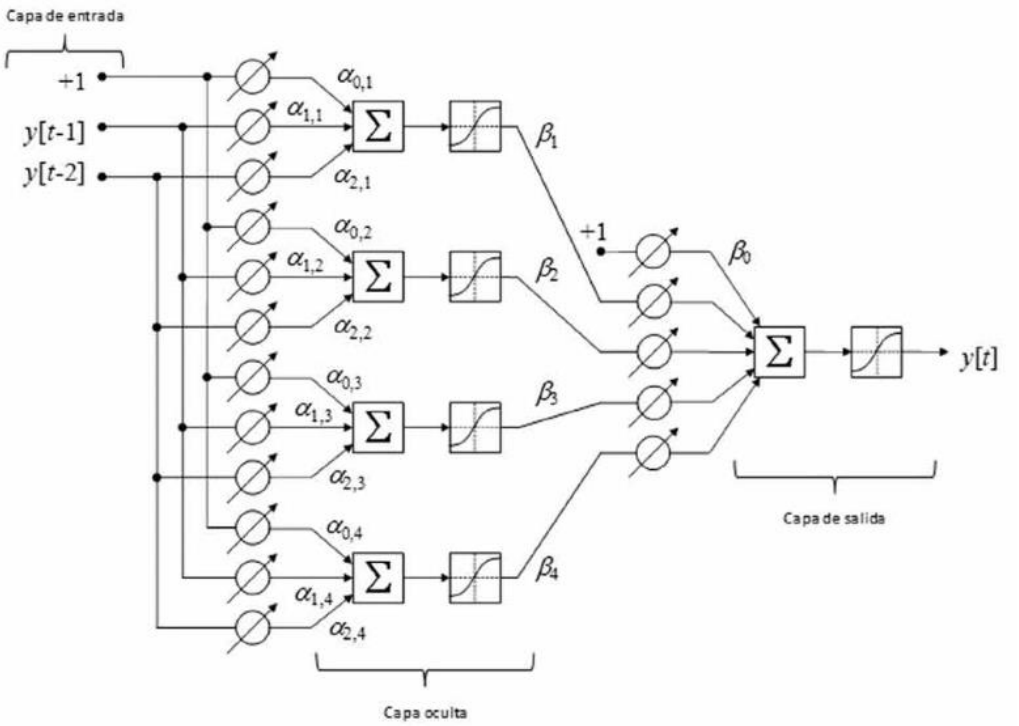


Figura 7 - Perceptrón Multicapa. [17]

4.3. Mecanismos de aprendizaje

Para poder obtener una salida basándose en las generalizaciones extraídas de los ejemplos anteriores de problemas del mismo tipo, es decir, en la experiencia, la red necesita pasar antes por un proceso de entrenamiento.

El proceso de aprendizaje no es más que modificar el peso sináptico de cada conexión en respuesta a una entrada determinada. La fuerza con la que se unen dos neuronas es emulada con el peso sináptico. Si el dicho peso tiene unos valores entre el rango [0,1], entonces un peso de uno significa una conexión muy fuerte y un peso cero es la destrucción o no conexión entre estas dos neuronas. El proceso de aprendizaje termina cuando todos los pesos sinápticos permanecen estables.

Dependiendo de si la red puede aprender online, durante su funcionamiento habitual, u off line, si ha de completar el aprendizaje antes de ponerse a funcionar, se distinguen dos métodos de aprendizaje: no supervisado en el primer caso y supervisado en el segundo [4]. Si se proyecta este concepto en la vida real, un ejemplo del aprendizaje supervisado sería el aprendizaje durante la niñez, cuando los adultos enseñan al niño si una acción es correcta o no se debe hacer (fase de entrenamiento). El niño crece con el código moral aprendido y sabrá distinguir acciones moralmente correctas de las que no lo son (fase de operación o funcionamiento). Y como ejemplo de aprendizaje no supervisado sería hacer un examen o los deberes, donde los estudiantes deben resolver los problemas extrayendo sus propias observaciones y conclusiones sin la ayuda de un supervisor que le diga la respuesta correcta.

5. Diseño y arquitectura de LEGION

Un aspecto fundamental en la percepción es la capacidad de poder unir espacialmente características sensoriales separadas para poder formar objetos, esencial para la identificación de objetos o figuras.

Wang [18] introdujo en 1995 el concepto de una red de osciladores inhibidor globalmente excitadores localmente (LEGION), inspirado en la teoría de correlación temporal: las células codificando diferentes características sensoriales en el cerebro son unidas si sus actividades temporales muestran una fuerte correlación. Y tradujo esta correlación temporal en correlación oscilatoria basándose en dos aspectos [19]:

- Sincronización dentro de un grupo de osciladores para representar un mismo objeto
- Desincronización entre diferentes grupos de osciladores para representar diferentes objetos

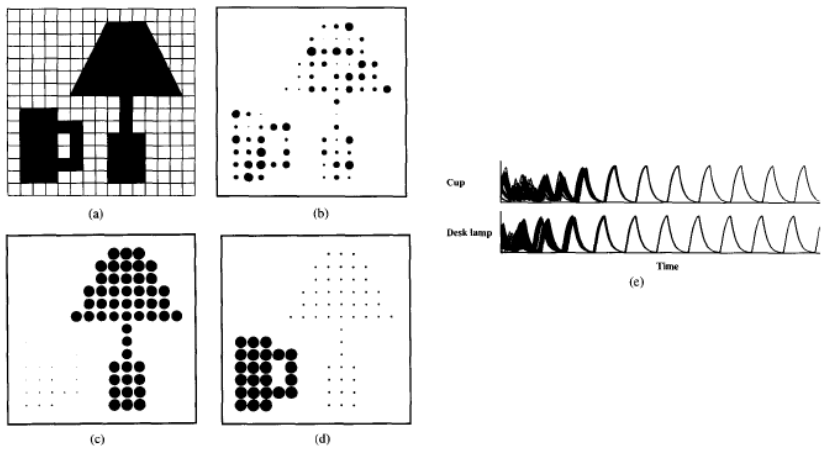


Figura 8 - Segmentación de patrones basando en la unión local de osciladores [19]
(a) imagen original (b) una captura instantánea de los osciladores al comienzo de evolución dinámica (c) y (d) captura de la actividad de los osciladores varios ciclos después (e) sincronización de los osciladores que pertenecen al grupo de la lámpara y al de la taza, y desincronización entre ambos grupos

Destaca decir que la red oscilatoria LEGION no emula una red neuronal genérica, sino una parte muy específica, el córtex visual humano. De ahí tiene unas aplicaciones muy concretas, concentradas en el campo de visión artificial.

Esta especialización le otorga a la red oscilatoria LEGION más ventajas con respecto a otros modelos de red:

- simplicidad organizacional, debido a que cada neurona u oscilador sólo está conectado con 4 osciladores vecinos. Esta característica hace que la red LEGION sea particularmente factible para una implementación de integración a muy gran escala (VLSI)
- sin necesidad de un mecanismo de aprendizaje para funcionar

5.1. LEGION: Principios de funcionamiento

En una red LEGION [18] de dos dimensiones (extensible a otras dimensiones), cada neurona u oscilador representa un píxel de la imagen y está conectada solo a cuatro de sus vecinos más próximos, que serían los osciladores de arriba, abajo, derecha e izquierda. Cuando reciben estímulos de entrada, los osciladores de la red empiezan a oscilar. Las oscilaciones no son aleatorias ni uniformes, sino que guardan una correlación con los datos de entrada (valor de cada pixel en la imagen). Así pues, cada objeto en la imagen es representado por un grupo de osciladores que oscilan en sincronismo. Diferentes objetos son representados por diferentes grupos de osciladores, y desincronizados entre cada grupo.

Además de la capa de osciladores, la red LEGION tiene otro oscilador llamado inhibidor global que está conectado con todos los osciladores de la red. Cuando el inhibidor recibe excitación de cualquier oscilador de la red, envía una señal inhibidora a toda red que reduce el potencial de todos los osciladores. Este mecanismo permite la desincronización de los diferentes grupos de oscilares, pudiendo así diferenciar distintos objetos en la imagen.

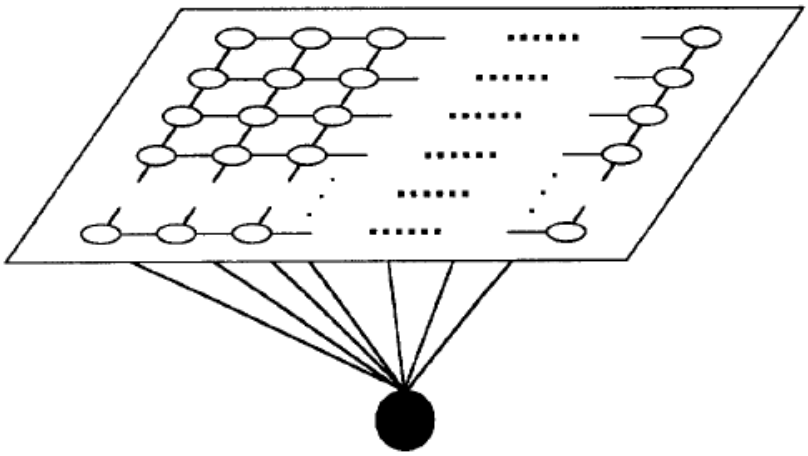


Figura 9- Arquitectura de una LEGION bidimensional [18]

Debido a dicha topología de LEGION, ésta adquiere una gran tolerancia a fallos. Si parte de la red es dañada, por ejemplo, de la mitad a la derecha, ésta puede seguir dando resultados de la parte que no está dañada, reconociendo figuras presentes en la otra mitad de la imagen.

5.2. Modelo de la neurona

En el artículo original [18], Wang definió el modelo de un oscilador como un lazo con realimentación entre una unidad excitadora (x_i) y una unidad inhibidora (y_i):

$$\frac{dx_i}{dt} = 3 \cdot x_i - {x_i}^3 + 2 - y_i + \rho + I_i + S_i \tag{5.1}$$

$$\frac{dy_i}{dt} = \epsilon[\gamma \cdot \left(1 + \tanh\left(\frac{x_i}{\beta}\right)\right) - y_i] \tag{5.2}$$

Donde x_i y y_i son potenciales de las unidades excitadora e inhibidora, ρ denota la amplitud del ruido gaussiano, I_i representa el estímulo externo al oscilador, $\{\epsilon, \gamma, \beta\}$ son parámetros, y S_i es la fuerza de unión con otros osciladores vecinos y se define como:

$$S_i = \sum_{k \in N(i)} W_{ik} \cdot S_\infty(x_k, \theta_x) - W_z \cdot S_\infty(z, \theta_{xz}) \tag{5.3}$$

Donde

$$S_\infty(x, \theta) = \frac{1}{1 + \exp[-K \cdot (x - \theta)]} \tag{5.4}$$

W_{ik} es el peso sináptico desde el oscilador k al oscilador i y N(i) es el conjunto de osciladores adyacentes conectados al i, K es un parámetro, θ_x es el umbral sobre el cual el oscilador puede afectar a sus vecinos, por último, W_z es el peso de la inhibición del inhibidor global z, cuya actividad se define como:

$$\frac{dz}{dt} = \Phi \cdot (\sigma_\infty - \mathrm{z}) \tag{5.5}$$

Aquí, si $x_i \geq \sigma_\infty \rightarrow z = 1$, es decir, si el potencial del excitador supera un umbral σ_∞, el inhibidor global enviará una señal de inhibición a toda red. En caso contrario, $x_i < \sigma_\infty \rightarrow z = 0$, el inhibidor global no actúa sobre la red. De esta forma, cada vez que la actividad de algún oscilador supere este umbral el inhibidor global actúa independientemente del número de osciladores que cumplen esta condición. El ritmo con el que el inhibidor global actúa sobre la red es controlado por el parámetro Φ.

Como se puede observar en las ecuaciones anteriores, si se implementa la red LEGION con el modelo del oscilador propuesto por el artículo original, resultaría demasiado costoso a nivel de hardware y sería interesante simplificar este modelo.

Bernard G. y Cesar T.H [20] propusieron un modelo modificado del oscilador basado en el modelo integrate-and-fire para su implementación en un FPGA.

5.2.1. Integrate-and-fire LEGION

Dicho oscilador basado en el modelo integrate-and-fire es descrito por:

$$\frac{dx_i}{dt} = -x_i + I_i - G + \sum_{j \in N(i)} \frac{\alpha_{ij}}{Z_i} \cdot P_j \tag{5.6}$$

Donde $N(i)$ son todos los osciladores vecinos del oscilador i, x_i es el potencial del dicho oscilador, Z_i es el número de vecinos que éste tiene, α_{ij} es la fuerza de unión, P_j es el estado de disparo del oscilador vecino, I_i es el estímulo externo que se aplica al oscilador y depende de la imagen de entrada, y por último, G es el impulso de inhibición instantáneo. Cuando el potencial supera a uno, $x_i \geq 1$, el oscilador se dispara y envía un potencial de acción a sus vecinos conectados; se vuelve a su potencial de reposo, $x_i \geq 0$, tras el disparo.

Esta ecuación guarda las características de la neurona integrate-and-fire al ser el potencial de la membrana una función integradora del tiempo. Y combina las tres ecuaciones del oscilador que propuso Wang en una única. El término $-x_i$ sería la unidad inhibidora del oscilador y x_i la unidad excitadora. La ecuación del oscilador inhibidor global se reduce a una señal G que se activa instantáneamente cuando un oscilador de la red genera un potencial de acción. Se ha mantenido el término del estímulo I_i y se ha incluido un nuevo término que calcula el peso sináptico neto que recibe de los osciladores vecinos con un sumatorio, basándose en su estado de disparo y la fuerza de unión con cada uno de ellos, α_{ij}, que se define como:

$$\alpha_{ij} = \begin{cases} 1, & |p_i - p_j| < umbral \\ 0, & |p_i - p_j| \geq umbral \end{cases} \tag{5.7}$$

Siendo p_i la intensidad del píxel que corresponde al oscilador i.

Con la ecuación 5.7 la fuerza de unión S_i se reduce a un valor binario, α_{ij}, que posteriormente contribuye al cálculo del peso sináptico neto o estímulo interno. Ahora, entre dos osciladores o hay unión ($\alpha_{ij} = 1$) o no la hay ($\alpha_{ij} = 0$), eliminando todas las posibilidades entre medio, como uniones débiles o fuertes. Esta comparación con un valor de umbral es llamada “pixel difference test” por los autores, donde se dice que un vecino j ha pasado el test si $\alpha_{ij} = 1$. El valor de umbral no es más que un parámetro de tolerancia: la diferencia en intensidad de pixel que se acepta como para considerar que dos pixeles de un color aproximado forman parte de una misma figura. Ya que, si la diferencia en color es muy grande, se percibe como dos figuras; y no todos los pixeles de la misma figura tienen la misma intensidad.

El valor del estímulo de entrada I_i depende también de esta fuerza de unión. Si ningún vecino pasa el test de diferencia de pixel, el valor $I_i = 0$ como es esperado, ya que estaría aislado y no recibe estímulos del vecindario. Si la mitad de sus vecinos pasan el test, $I_i = 1{,}25$ y en los demás casos $I_i = 0{,}95$.

$$I_i = \begin{cases} 0, & \frac{1}{Z_i} \cdot \sum_{j \in N(i)} \alpha_{ij} = 0 \\ 1{,}25, & \frac{1}{Z_i} \cdot \sum_{j \in N(i)} \alpha_{ij} = 0{,}5 \\ 0{,}95, & en\ los\ demás\ casos \end{cases} \tag{5.8}$$

Estos tres valores son particularmente interesantes, pues asignan a cada oscilador una función, por el orden que aparecen en la ecuación 5.8 se clasifican en:

- osciladores aislados que no forman parte de ninguna figura
- osciladores líderes que lideran el primer impulso en cada figura
- osciladores que forman parte de la figura, pero sus disparos son arrastrados por los osciladores líderes

Para evitar una complejidad de implementación innecesaria con operaciones no lineares, se resuelve la ecuación diferencial 5.6 en una ecuación de diferencias utilizando el método de Euler [20].

$$x_i[n+1] = x_i[n] + \frac{1}{h} \cdot (-x_i[n] + I_i - G + \sum_{j \in N(i)} \frac{\alpha_{ij}}{Z_i} \cdot P_j\ ) \tag{5.9}$$

Siendo $x_i[n+1]$ el potencial del oscilador en el siguiente intervalo de tiempo, $x_i[n]$ en el presente intervalo de tiempo y $1/h$ ancho del intervalo.

Con estas modificaciones del modelo del oscilador, la red que se va a implementar no será considerada integradamente LEGION, al ser 'Integrate-and-Fire' un modelo de neurona de redes SNN (Spiking Neuronal Network). De esta manera, la LEGION modificada combina características de ambas redes, pero por comodidad, se llamará LEGION de aquí en adelante.

5.2.2. Adaptaciones del modelo

Para la ecuación de 5.9, se ha escogido empíricamente un ancho de intervalo de integración $\frac{1}{h} = \frac{1}{32}$. Se ha observado que, a un ancho muy grande, por ejemplo $\frac{1}{h} = \frac{1}{1} = 1$, todos los osciladores líderes disparan a la vez sin importar a qué grupo o figura corresponden y a una frecuencia de cada dos ciclos de reloj, uno en el que el estado de disparo está a on y otro reseteado, sin que el inhibidor global tenga

efecto alguno en la red. Esto es debido a que todos los líderes tienen una entrada de estímulo $I_i = 1{,}25$ y dispararían siempre al ser este estímulo siempre superior al umbral de disparo. Si se escoge un ancho de intervalo demasiado pequeño, los osciladores líderes tardarían demasiado en disparar y disminuiría la eficiencia de la red.

Además, para el denominador h se ha escogido un número que es potencia de 2, para un mínimo coste de implementación en hardware. Pues la división en binario es una operación que requiere muchos recursos, pero si este número es potencia de 2, $h = 2^s$, la división sería tan sencilla como un desplazamiento aritmético hacia la derecha de s bits.

Aunque este modelo de Bernard G. y Cesar T. H. simplifica en gran medida las ecuaciones del modelo original y lo hace más apto para implementación en hardware, se ha de hacer unas pequeñas modificaciones para adaptarlo al ancho de intervalo h de la nueva ecuación de diferencias.

La ecuación adaptada quedaría:

$$x_i[n+1] = x_i[n] + \frac{1}{32} \cdot (-x_i[n] + I_i - G \cdot G_{value} + w_{value} \cdot \sum_{j \in N(i)} \frac{\alpha_{ij}}{Z_i} \cdot P_j \ + x_{ruido}) \qquad (5.10)$$

En la nueva ecuación adaptada, el potencial del oscilador se reduce en una cantidad fijada por G_{value} y aumenta más rápidamente por la presencia del coeficiente w_{value}. Estos dos parámetros se han hallado empíricamente al observar que sin G_{value} el inhibidor global reduce en gran medida el potencial de los osciladores y no permitiría que los no líderes puedan saltar; y sin w_{value}, los osciladores no líderes tampoco pueden disparar debido a que con un ancho de intervalo 1/32, el peso sináptico que reciben en cada ciclo de reloj es demasiado pequeño en comparación con la señal de inhibición global.

Se ha añadido un término de ruido a la ecuación, considerando que, en una situación real, la transmisión de señales tanto entre osciladores como en las señales de entrada no es perfecta y puede haber interferencias.

Otras modificaciones que no reflejan en la ecuación 5.10 se explicarán en su apartado correspondiente en el punto 5.2.3.

5.2.3. Diseño y arquitectura de la neurona

Como la LEGION es una red con una aplicación muy específica, segmentación de figuras, su topología está diseñada para este fin. Por este motivo, para la implementación de esta red en este proyecto también se ha considerado este aspecto desde el inicio de la fase de diseño. Se define desde el principio que la red trabajará con imágenes en escala de grises, ya que para la mayoría de las aplicaciones

industriales una imagen en color no aporta una cantidad de información suficientemente mayor como para compensar el coste de implementación.

Para poder realizar la suma de todos los términos y compararla con el umbral de disparo, se ha de calcular previamente el valor de cada término. Para ello, se ha diseñado una serie de unidades que compone el oscilador.

5.2.3.1. Pixel Difference Test

Esta unidad realiza la función del test de diferencia de pixeles hablada anteriormente y devuelve el valor de la fuerza de unión (α_{ij}) de cada oscilador con sus vecinos. Recordando que

$$\alpha_{ij} = \begin{cases} 1, & |p_i - p_j| < umbral \\ 0, & |p_i - p_j| \geq umbral \end{cases} \tag{5.11}$$

Antes de empezar a diseñar esta unidad, hay que tener presente que cada oscilador representa un pixel de la imagen, por lo que el tamaño de la red depende del tamaño de la imagen a analizar. Debido a esta característica, la distribución de red es siempre un rectángulo (o cuadrado) con osciladores interconectados como se observa en la Figura 10.

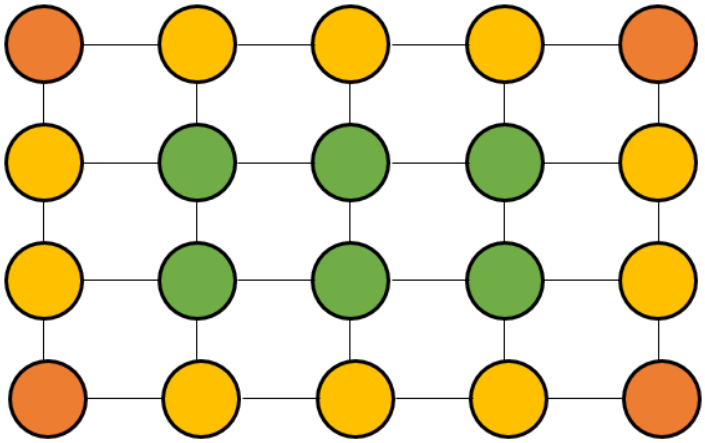


Figura 10 - Distribución de los osciladores en una red LEGION bidimensional de 5x4

De esta forma, los osciladores de la esquina de la imagen (en marrón) tienen solo dos vecinos conectados, los osciladores del borde (en amarillo) tienen tres y los del centro de la imagen, 4. La suma de las fuerzas de unión es dividida después por el número de vecinos (Z_i) que tiene ($\frac{1}{Z_i} \cdot \sum_{j \in N(i)} \alpha_{ij}$), garantizando de esta forma que todos los osciladores, esté en el interior o en los bordes, reciben un peso sináptico efectivo neto equivalente al número de vecinos que tiene [18], [20].

Dicho esto, la unidad Pixel Difference Test (PDT) tiene la siguiente arquitectura diseñada:

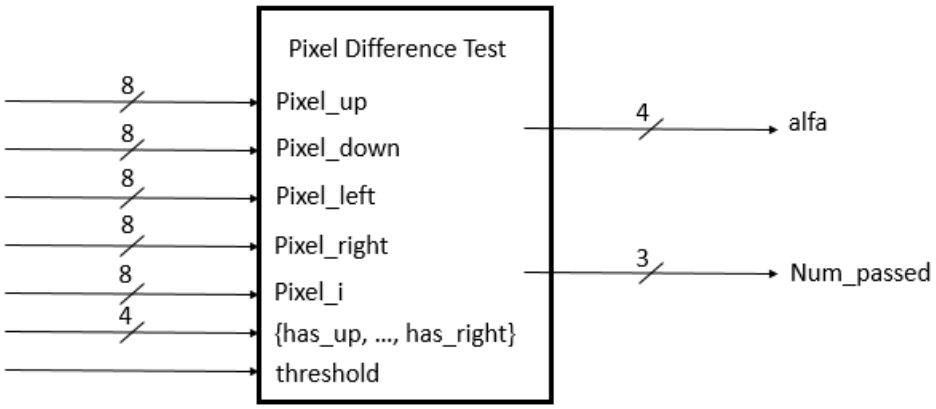


Figura 11 - unidad PDT

Donde vector de salida *alfa* es un vector de 4 bits, en donde la posición de cada bit representa la posición del vecino correspondiente (ver Figura 12). Así pues, si el bit cero tiene el valor cero, significa que el oscilador que está encima del oscilador i en cuestión no ha pasado el test y, por lo tanto, están desconectados entre sí.

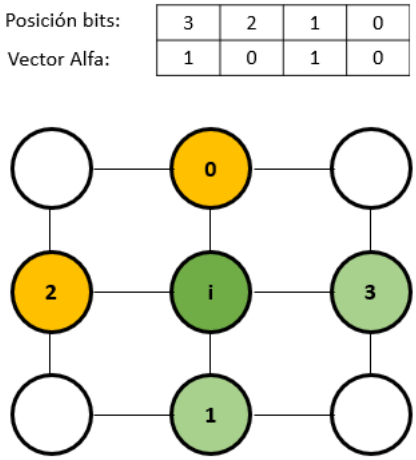


Figura 12 - Relación entre la posición de los bits del vector Alfa con la posición relativa en la red con el oscilador i. En verde claro: vecinos conectados del oscilador i. En amarillo: vecinos no conectados del oscilador i.

Como los osciladores de los bordes no llegan a tener 4 vecinos, se introducen 4 parámetros {*has_up, has_down, has_left, has_right*} para forzar el bit de la posición correspondiente del vector *alfa* a cero.

Se ha definido que la intensidad de pixel se expresa con un vector de 8 bits. Y basándose en que un vector de 8 bits tomaría valores entre [0, 255], se ha establecido un umbral de valor 16, que sería 16/255 del valor máximo o 6,25% de diferencia en la intensidad de pixel. Como se puede apreciar en la Figura 13, una diferencia de 6% en la escala de grises es casi imperceptible por el ojo humano.

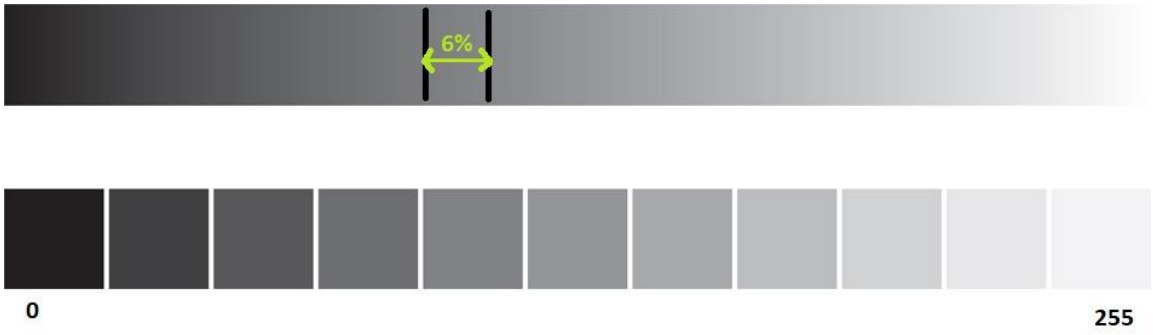


Figura 13 - Escala de grises. Adaptación de la imagen en [21]

La estructura interna de la unidad *pixel difference test* es:

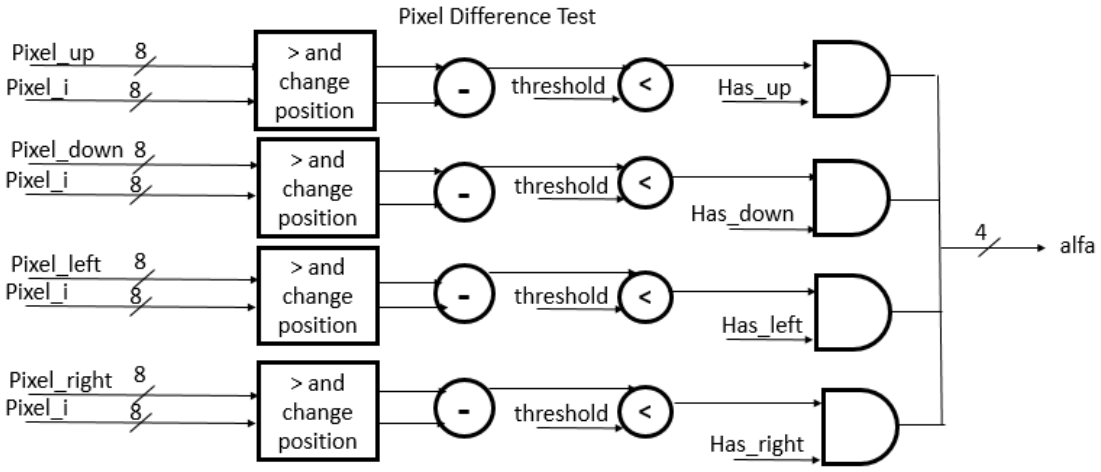


Figura 14 - Estructura interna de la unidad PDT

Para evitar trabajar con el valor absoluto, se comparan los valores de la intensidad de pixel del oscilador i y su vecino j. El valor mayor pasa a ser minuendo y el menor, sustraendo. Después de la resta, se hace comparar la diferencia con el valor umbral. Si el resultado es positivo y este vecino existe en la red, se pasará el test y habrá conexión entre estos dos osciladores. Por este motivo, a partir de este punto se utilizará indistintamente "vecinos conectados" y "vecinos que pasan el test". Hay que tener presente también que, no es lo mismo el número de vecinos (Z_i) que el número de vecinos conectados ($\sum_{j \in N(i)} \alpha_{ij}$). El número de vecinos es determinado por la construcción de la red, depende solo del tamaño de la red. Mientras que el número de vecinos conectados depende de la imagen a analizar, y tomará un valor nunca superior al número de vecinos.

5.2.3.2. Leader detecter

Un oscilador es leader si su valor de estímulo de entrada es mayor que la unidad.

$$I_i = \begin{cases} 0, & si\ \frac{1}{Z_i} \cdot \sum_{j \in N(i)} \alpha_{ij} = 0 \\ 1{,}25, & si\ \frac{1}{Z_i} \cdot \sum_{j \in N(i)} \alpha_{ij} = 0{,}5 \\ 0{,}95, & en\ los\ demás\ casos \end{cases} \quad (5.12)$$

En la unidad PDT se provecha el vector Alfa para obtener el sumatorio ($\sum_{j \in N(i)} \alpha_{ij} = num\_passed$) de fuerzas de unión, simplemente sumando todos los bits del vector:

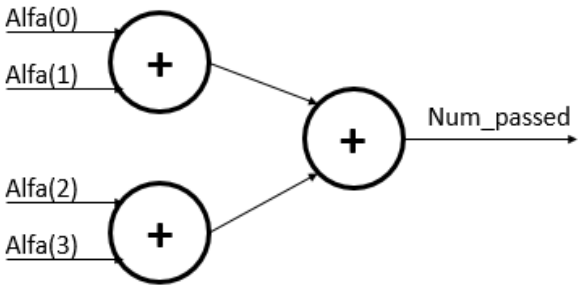


Figura 15 - Operación de suma de todos los bits del vector Alfa para obtener el número de osciladores que han pasado el test

Una vez que se ha obtenido el número de osciladores con los que el oscilador *i* puede tener conexión, se podrá saber si el oscilador i es líder o no dependiendo de su número de vecinos de esta forma:

$$líder\ si \begin{cases} \sum_{j \in N(i)} \alpha_{ij} = 2, & si\ Z_i = 4 \\ \sum_{j \in N(i)} \alpha_{ij} = 1, & si\ Z_i = 2 \\ \sum_{j \in N(i)} \alpha_{ij} = 1{,}5, & si\ Z_i = 3 \end{cases} \quad (5.13)$$

Sin embargo, este criterio basándose en el número de vecinos no es ideal, por los siguientes problemas:

- En un oscilador del borde (con tres vecinos), ¿se le considera líder con dos vecinos conectados o con uno? Ya que no se puede tener 1,5 vecinos.
- Se ha observado empíricamente que si la figura que forman los osciladores del interior (con cuatro vecinos) es un rectángulo o un cuadrado, solo habrá 4 líderes para esta figura. Y no aportarán fuerza suficiente para que todos los osciladores de la misma figura puedan disparar. Como se puede apreciar en la Figura 16, los líderes (en verde) disparan y aportan un estímulo para que sus vecinos (en color carne) puedan disparar también, pero los más interiores (en

gris) recibirán 2 o más ciclos de reloj de señal de inhibición global antes de recibir el estímulo del vecino, que reducirá el potencial en mayor medida que el aumento de potencial aportado por dicho estímulo. Puede que este criterio del líder es ideal para otro ancho de intervalo ($1/h$) escogido.

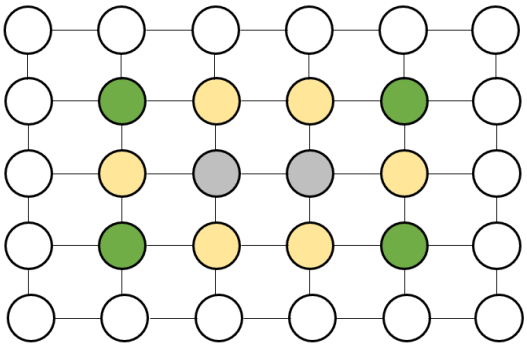


Figura 16 - Ejemplo de un rectángulo en una LEGION con el criterio del líder de Bernard G.

Después de simular la red con diferentes criterios de líder, se ha encontrado un criterio que mejor resultado da con la ecuación adaptada. El nuevo criterio consiste en considerar líderes todos los osciladores del contorno de una figura. Abstrayendo una figura cualquiera:

- Si el oscilador tiene 4 vecinos, no puede ser contorno de una figura si está conectado con todos los vecinos. El número de vecinos conectados para ser contorno es, por lo tanto, menor que 4.
- Un oscilador del borde de la imagen (con 2 ó 3 vecinos) es contorno de una figura por definición siempre que el número de vecinos conectados sea no nulo, ya que nunca pueden estar en el interior de una figura.
- Independientemente del número de vecinos, si el oscilador solo tiene un vecino conectado, éste puede formar parte del contorno o parte de una línea aislada que se extiende de la figura. No es necesario hacerlo líder porque puede ser estimulado por sus vecinos, líderes o vecinos de líderes.

En conclusión, un oscilador puede ser contorno de una figura, y, por lo tanto, líder, si tiene 2 ó 3 vecinos conectados, independientemente del número de vecinos que tiene.

Basándose en este nuevo criterio, la unidad Detector de Líder (Leader Detecter) es un simple multiplexor con un comparador.

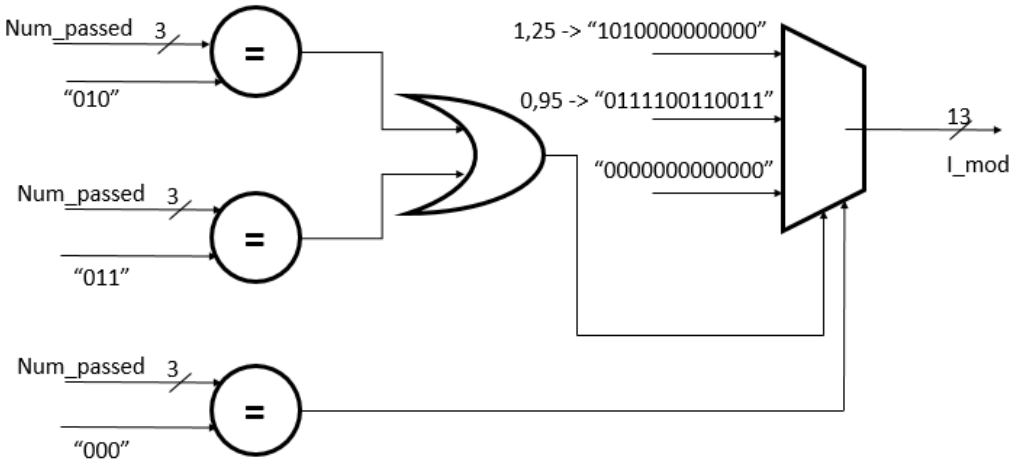


Figura 17 - estructura interna de la unidad Leader Detecter

5.2.3.3. LFSR

Se añade un término de ruido a la ecuación para proporcionar mayor realismo y comprobar así, su tolerancia a interferencias. Para realizar este término, se necesitaría un generador de número arbitrario. Para aproximar lo más posible a un generador de número arbitrario y buscando una solución que no añade demasiada complejidad al sistema, se dio como solución el registro de desplazamiento con realimentación lineal (Linear Feedback Shift Register, LSFR).

Dependiendo del número (n) de biestables de tipo D encascados, se podrá generar $(2^n - 1)$ números que, siendo el algoritmo determinista, aparecerán como aleatorios a simple vista al no seguir un orden lineal. Al terminar todo el ciclo, el valor del registro se resetea al valor inicial o semilla, y vuelve a empezar siguiendo la misma secuencia.

Si se escoge un LFSR de 8 bits, éste podrá generar 255 valores diferentes en cada ciclo y el polinomio que lo define tiene la siguiente expresión [22]:

$$x^8 + x^6 + x^5 + x^4 + 1 \tag{5.14}$$

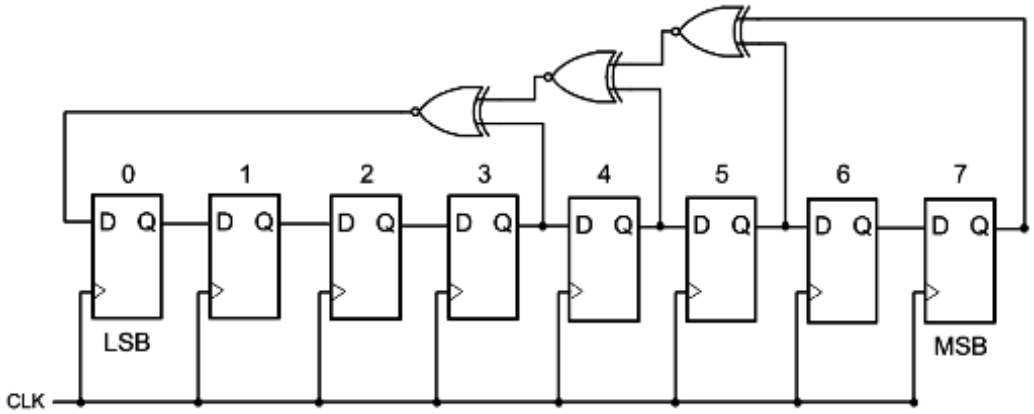


Figura 18- Arquitectura de un LFSR de 8 bits [23]

5.2.3.4. Arithmetic Unit

Cuando se tiene la mayoría de los términos de la ecuación, la unidad aritmética se encarga de calcular el término del peso sináptico neto y la suma neta de todos estos términos. Recordando que:

$$x_i[n+1] = x_i[n] + \frac{1}{32} \cdot (-x_i[n] + I_i - G \cdot G_{value} + w_{value} \cdot \sum_{j \in N(i)} \frac{\alpha_{ij}}{Z_i} \cdot P_j \ + x_{ruido}) \quad (5.10)$$

Para realizar el término del peso sináptico se hace pasar por una puerta AND la señal del disparo y la fuerza de unión del oscilador vecino correspondiente. De esta forma se puede contar el número de vecinos conectados que están activos. Dependiendo de este número y del número total de vecinos que tiene, se le asignará un valor de peso sináptico. Este valor se ha calculado previamente teniendo en cuenta todas las combinaciones de la suma de impulsos recibidos y el número de vecinos, que posteriormente se ha multiplicado por el coeficiente w_value y dividido por el ancho de intervalo 1/32, para evitar hacer divisiones y multiplicaciones en hardware. En código VHDL sería:

```
weight_shifted <= "0000000000000" when (sum_P = 0) else                                    -- 0
                  "0100000000000" when (Zi = 2 and sum_P = 1) or (Zi = 4 and sum_P = 2) else   -- 0.5
                  "0010101010101" when (Zi = 3 and sum_P = 1) else                         -- 0.33
                  "0101010101011" when (Zi = 3 and sum_P = 2) else                         -- 0.66
                  "0010000000000" when (Zi = 4 and sum_P = 1) else                         -- 0.25
                  "0110000000000" when (Zi = 4 and sum_P = 3) else                         -- 0.75
                  "1000000000000";
```

donde *weight_shifted* es el peso sináptico neto ya dividido por el ancho de intervalo, *sum_p* es el número de vecinos conectados y *Zi* es el número de vecinos.

Por otro lado, con un multiplexor se elegirá un valor G_value o un valor nulo con la señal de inhibición global. Teniendo todos los términos listos, se suman por un lado todos los términos positivos, y por el otro lado, los negativos. Restándolos se obtiene el valor del potencial del oscilador para la siguiente iteración.

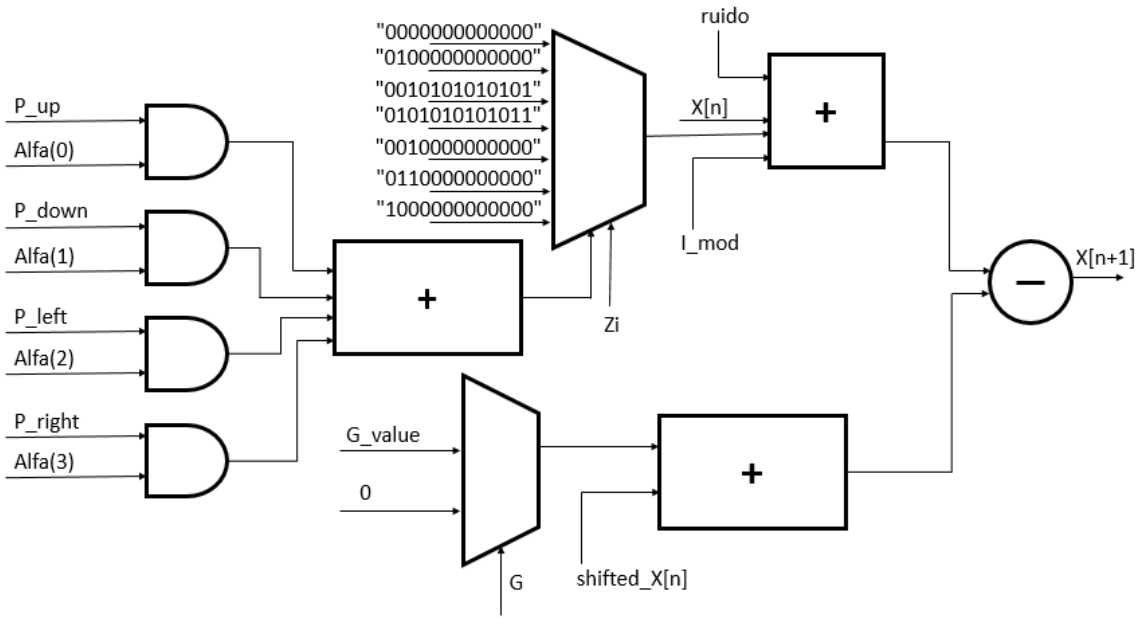


Figura 19 - Arquitectura de la unidad aritmética

5.2.3.5. RAM

Todas las unidades que compone el oscilador, excepto el generador de ruido LFSR, son circuitos combinacionales. Para poder actualizar el valor del potencial para la siguiente iteración se necesita un registro a la salida de la unidad aritmética. Este valor solo se actualiza con el flanco de subida del reloj, manteniendo y suministrando el valor anterior a la unidad aritmética, por lo que en alguna medida se parece a una memoria RAM.

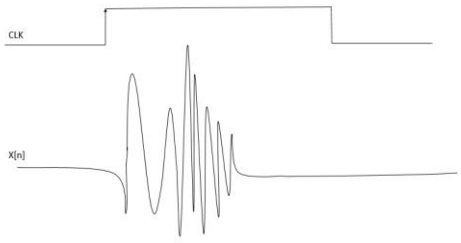


Figura 20 - Comportamiento del potencial del oscilador

Como se observa en la Figura 20, en el flanco de subida del reloj, el oscilador recibe nuevos valores del potencial de la iteración anterior, del ruido y de los estados de disparo de sus vecinos, siendo los demás términos constantes si la imagen de entrada es la misma. La unidad RAM no guardará ningún valor del potencial en estos momentos, sino solo en el flanco de subida del reloj, cuando el bloque combinacional ya se ha estabilizado y ha dado el resultado de la suma.

Si el valor a la salida de RAM es mayor que 1, el oscilador generará un impulso. Cabe mencionar que, para dar mayor precisión a la ecuación del oscilador, el potencial es un vector de 12 bits de punto fijo con rango de valores [0, 4095] que representa al rango [0, 1]. La conversión se hace de la siguiente manera:

$$x_i[0, 4095] = valor\ decimal\ a\ representar \cdot 2^{12} \tag{5.15}$$

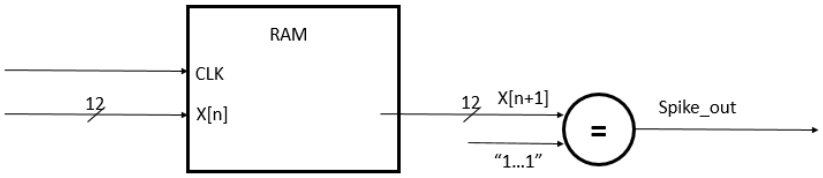


Figura 21 - Arquitectura de la unidad RAM y el generador de impulsos

5.2.3.6. Unidad neurona

Uniendo todas estas unidades, se obtiene la unidad básica de la red, la neurona o el oscilador, que tiene la siguiente estructura:

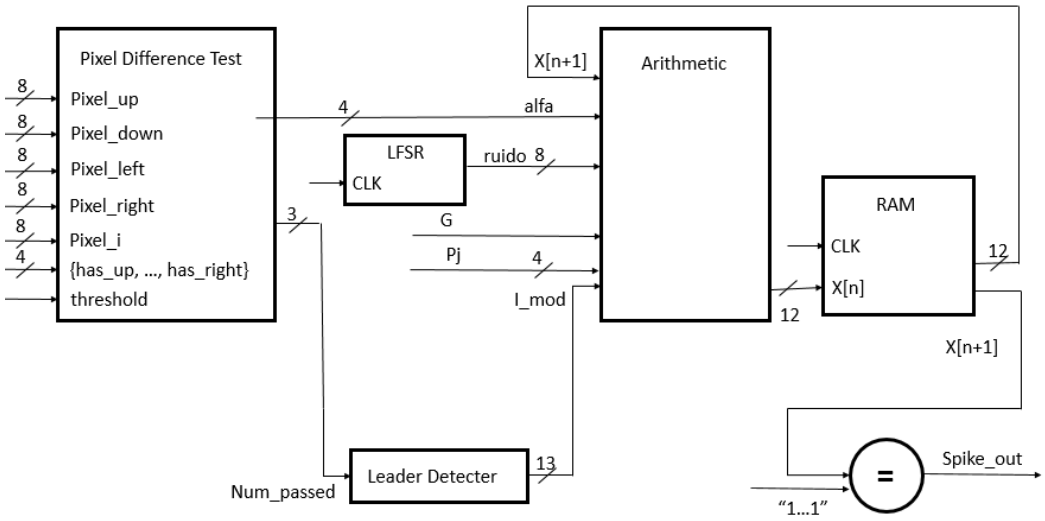


Figura 22 - Arquitectura de un oscilador

5.3. Red neuronal LEGION

Como ilustra la Figura 3, existen 3 niveles o capas en una red neuronal artificial. La red LEGION puede distribuirse en los tres niveles de la siguiente forma:

- Una capa de entrada donde cada neurona lee la intensidad del pixel de la imagen de entrada y la transforma en un vector de 8 bits que envía al oscilador correspondiente en la capa oculta
- La 1ª capa oculta la comprenden los osciladores y la 2ª, el inhibidor global, conectado con todos los osciladores de la primera capa oculta
- La capa de salida la constituyen el generador de impulsos de cada uno de los osciladores y unos circuitos electrónicos que adaptan estos impulsos a unos equipos de salida, ya que un equipo de salida, por ejemplo, un monitor, no es capaz de leer directamente los impulsos de los osciladores

En este apartado solo se hablará de las capas ocultas, dejando las otras dos en el apartado Implementación de LEGION en FPGA.

5.3.1. Globally Inhibitory Unit

El inhibidor global no es más que un comparador, si algún oscilador está activo generará una señal de inhibición. Aunque el estado del impulso o spike de los osciladores es un valor binario, al ser la red de un tamaño mayor que 100 osciladores en la capa oculta, el inhibidor global tendría una cantidad inmensa de entradas. Para ello, se combinan el estado de impulso de todos los osciladores de la red en un vector donde la posición del bit en el vector indica también la posición del oscilador en la red.

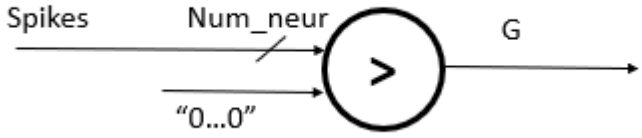
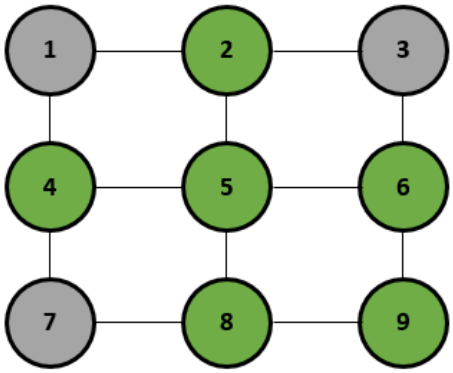


Figura 23 - Arquitectura del inhibidor global

Como se observa en la Figura 24, un ejemplo de una red 3x3, el estado de disparo (en verde si dispara, y en gris lo contrario) de cada oscilador de la red se refleja en el vector Spikes, siguiendo un orden de enumeración de izquierda a derecha y de arriba abajo.



Posición	9	8	7	6	5	4	3	2	1
Vector spikes	1	1	0	1	1	1	0	1	0

Figura 24 - Correspondencia de la posición del oscilador en la red con la del vector Spikes

5.3.2. Diseño y arquitectura de la LEGION

Para la interconexión de la red, solo importan los impulsos de los osciladores y la señal de inhibición global. Las demás entradas son fijadas o por construcción o por la imagen de entrada, que manteniéndolos constantes, se puede ver la unidad oscilador como el bloque de la derecha en la Figura 25. No se trata de eliminar estas entradas, sino una representación simplificada para introducir la interconexión de varios osciladores.

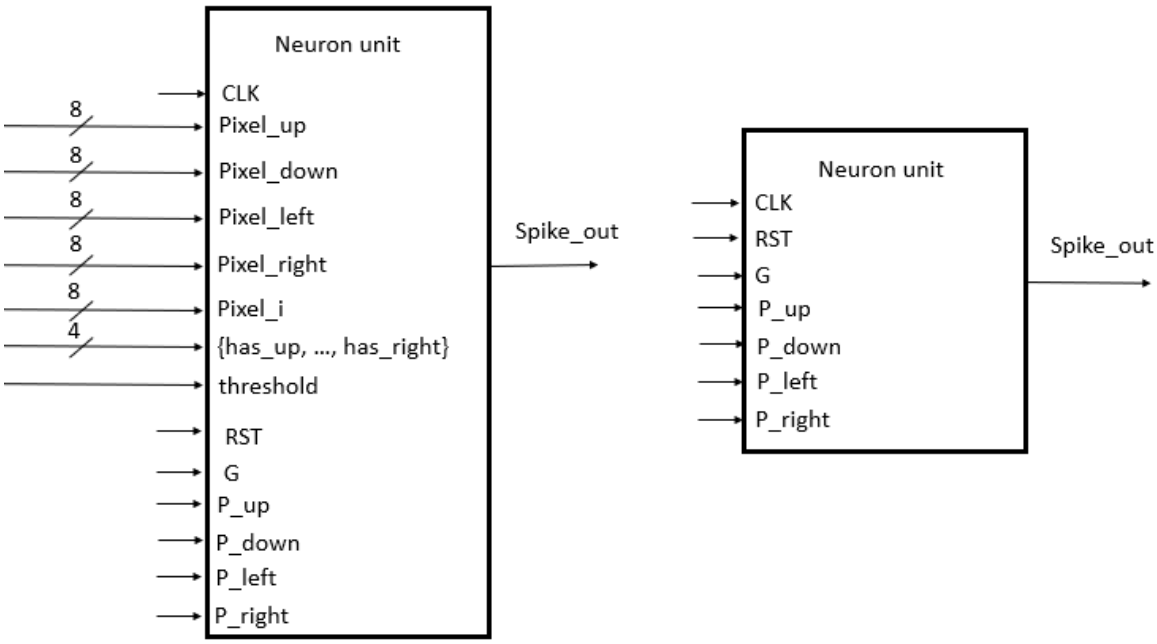


Figura 25 - Arquitectura general del oscilador y su versión reducida

Como modo de ejemplo, se muestra en la Figura 26 la interconexión de 4 osciladores en una red de 3x3. Los impulsos de salida de los osciladores se conectan a la entrada correspondiente de sus vecinos. A los osciladores del borde de la imagen se les asigna un valor nulo para aquellos valores que corresponden a un vecino inexistente: el valor de pixel y el estado de disparo.

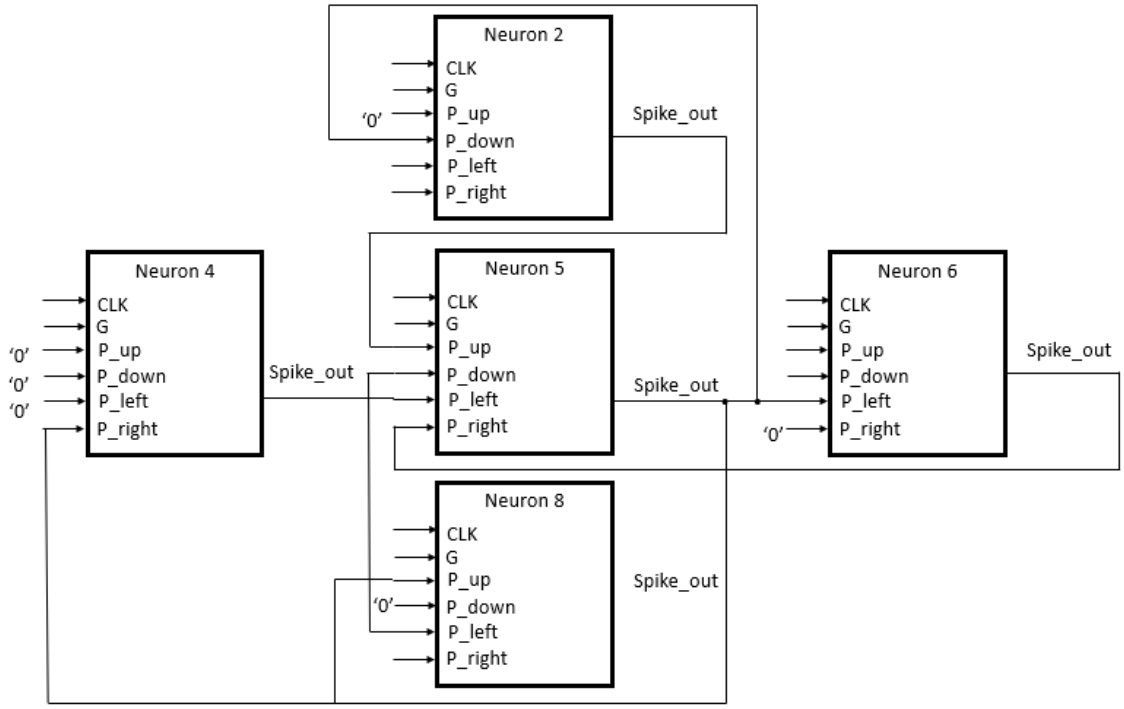


Figura 26 - Ejemplo de conexión entre osciladores en una red de 3x3

5.3.3. Mecanismos de reset

Cuando se cambia de imagen de entrada, el potencial de los osciladores empezará a evolucionar partiendo de un potencial residual. Este potencial residual puede tener un valor entre [0, 4094] ó [0, 0,9995] si se hace la conversión, y puede afectar gravemente al funcionamiento de la red. Por lo tanto, se debe hacer un reset cada vez que se ha cambiado de imagen.

Una opción de reset sería poner a cero todas las variables implicadas en la suma. Al tener tres sumandos (en el código VHDL), se tendría que crear un multiplexor para cada bit del vector, para 3 vectores, y para todos los osciladores de la red.

Por lo tanto, se ha optado por otra opción que utilizase menos recursos: imagen de reset (ver Figura 28) y señal de *noise off*. Este método consiste en enseñar a la red una imagen en la que ningún pixel pasa el "pixel difference test", es decir, ningún oscilador en la red tiene vecinos conectados. De esta forma, el estímulo de entrada es nulo y, por consiguiente, el potencial será nulo siempre que el término ruido es desactivado.

Recordando la ecuación que define el potencial de un oscilador:

$$x_i[n+1] = x_i[n] + \frac{1}{32} \cdot (-x_i[n] + I_i - G \cdot G_{value} + w_{value} \cdot \sum_{j \in N(i)} \frac{\alpha_{ij}}{Z_i} \cdot P_j + x_{ruido}) \quad (5.10)$$

Al no tener ningún vecino conectado, el estímulo de entrada I_i y el peso sináptico neto $w_{value} \cdot \sum_{j \in N(i)} \frac{\alpha_{ij}}{Z_i} \cdot P_j$ son nulos. Desactivando el término ruido, el potencial del oscilador decrecería bruscamente y se estabiliza a un valor muy próximo a cero, como se puede observar en las simulaciones de la Figura 27.

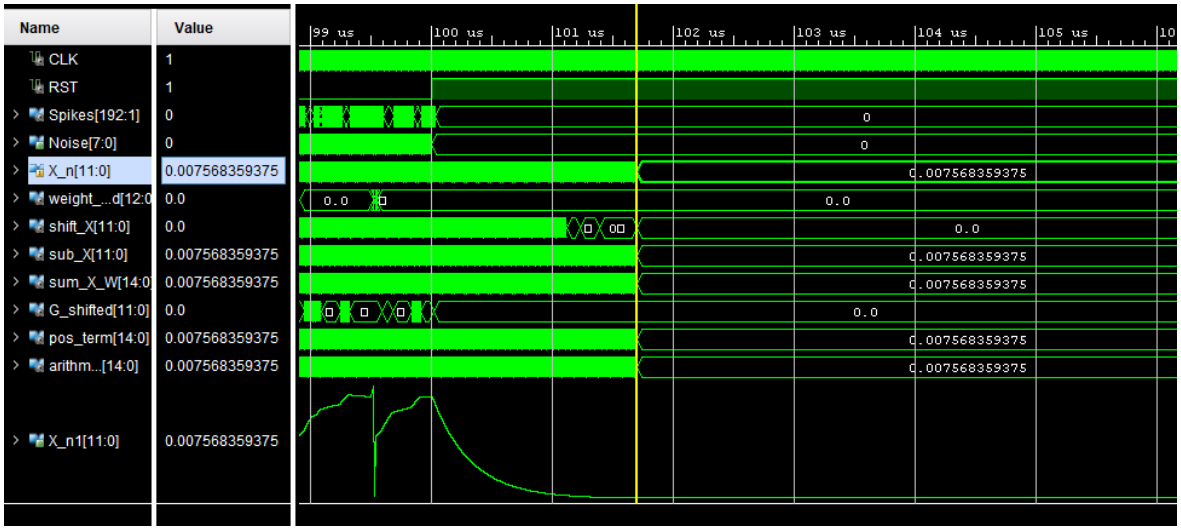


Figura 27 - Evolución del potencial con imagen de reset y señal de desactivación de ruido

Con la imagen de reset y la señal (asignada a un interruptor llamado RST) que desactiva el ruido, todos los términos menos el potencial se resetean. En menos de dos microsegundos, el potencial y todas las variables internas de suma se estabilizan en un valor próximo a cero, 0,007.

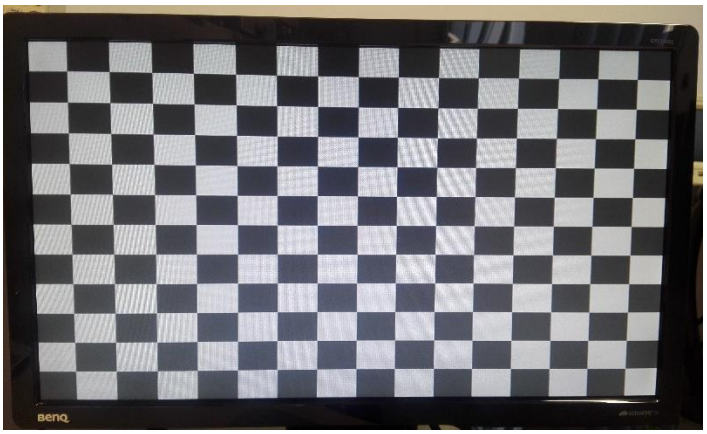


Figura 28 - Imagen de entrada a la red con función de reset

5.3.4. Adaptaciones del diseño para la implementación

En las simulaciones se puede comprobar el comportamiento del sistema. Sin embargo, para poder implementar con éxito el sistema diseñado en el FPGA, se ha cumplir con los tiempos de sincronización o timing. En un datapath, la señal eléctrica tarda un cierto tiempo en recorrer desde el origen hasta el destino asignado. Al ser el sistema síncrono, la recorrida de un datapath debe durar menos de un ciclo de reloj, 10 ns. De no ser así, la información no será sincronizada y el sistema será una caja negra cuyo comportamiento no es predecible.

Para poder implementar el sistema con éxito, se ha de resolver los problemas de timing. Éstos se concentran en unos pocos datapaths que duraban más de 10 ns: los que recorrían desde la salida de RAM hasta la entrada de RAM, pasando por la unidad aritmética; y los que unían de un bloque al otro.

Se han encontrado tres soluciones para resolver los problemas de timing:

a) Fast Slew Rate Outputs

Las Fast Slew Rate Outputs son salidas más rápidas que una salida convencional porque conducen una carga de impedancia reducida [24]. De esta forma, se puede reducir unos pocos nanosegundos en el datapath. Para que una salida sea de fast slew rate, se ha de definir en el archivo constraints de la siguiente forma:

```
set_property SLEW FAST [get_ports {spikes_line[0]}]
```

b) Registros en la salida

Las Fast Slew Rate Outputs son utilizadas solo para señales de salida al sistema y no pueden ser aplicadas a salidas de un bloque dentro del sistema. Si no existe realimentación entre dos bloques, en donde la salida de un bloque es entrada del otro bloque, se puede aplicar registros entre estos bloques. Esta limitación de aplicación es debida a que la señal tarda un ciclo de reloj más en llegar al siguiente bloque, e influiría en el comportamiento de sistemas con realimentación.

Estos registros acortan la distancia que la señal debe recorrer en 10 ns. Si una señal antes ha de recorrer dos bloques enteros en 10 ns, ahora solo ha de recorrer un bloque en el mismo tiempo, reduciendo considerablemente el WTS (Worst Negative Slack) o el retraso.

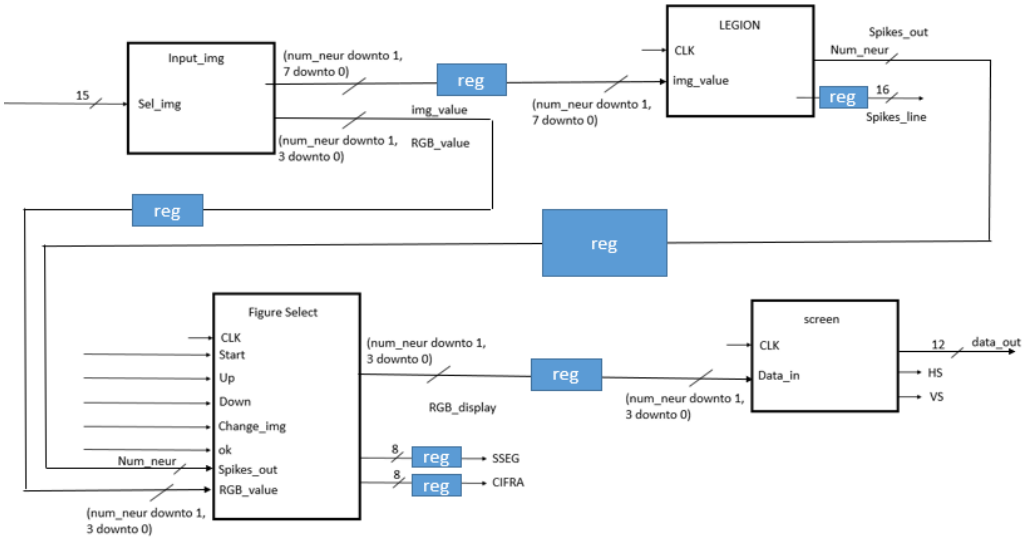


Figura 29 - Aplicación de registros de salida en el sistema diseñado

c) Técnicas de optimización

Después de aplicar las dos soluciones anteriormente mencionadas, se solucionaba la mayoría de los problemas de timing. Sin embargo, el bloque LEGION contiene una cantidad inmensa de circuitos combinacionales, y se ha de buscar otros métodos para reducir el WNS.

Según los path reports de Vivado, los problemas de timing se concentran en la suma de todos los términos de la ecuación, al generar una cantidad inmensa de operaciones de carry. La solución fue en hacer sumas paralelas en lugar de sumas en serie, reduciendo el tiempo total en acabar todas las operaciones.

Como se puede observar en la Figura 30, si una operación de suma tarda x ns, se tardan 4x ns antes de optimizar y 3x ns una vez optimizada, manteniendo el mismo uso de recursos.

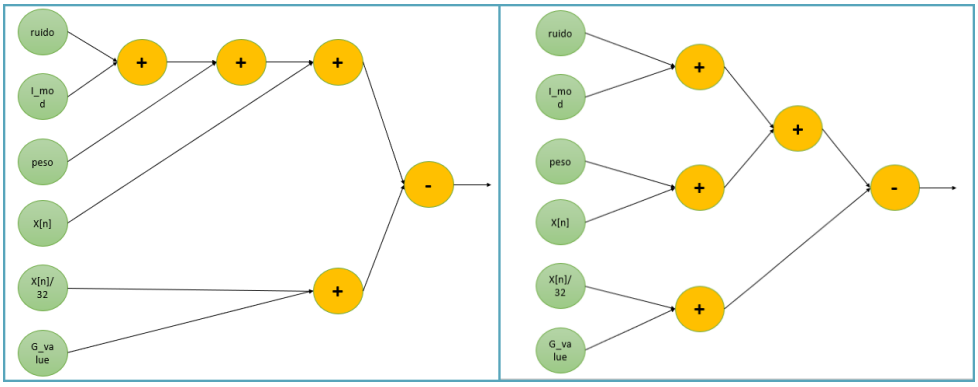


Figura 30 - Operaciones de suma antes (izquierda) y después (derecha) de optimización

Tras aplicar estas tres técnicas de reducción de retraso, se pudo implementar con éxito el diseño en el FPGA.

5.3.5. Simulaciones y resultados experimentales

Para mostrar el comportamiento de los osciladores, se ha simulado el modelo del oscilador y de LEGION con el programa Vivado y con un osciloscopio Tektronix de modelo MDO3034 para visualizar los resultados de la implementación en el dispositivo FPGA.

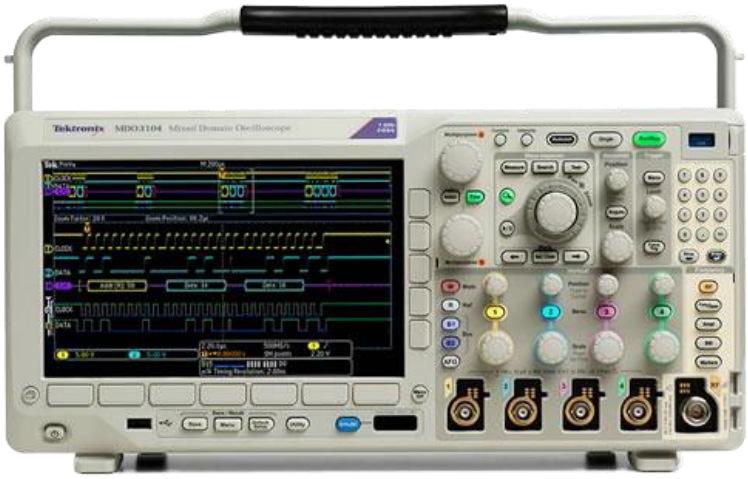


Figura 31 Osciloscopio Tektronix MOD3034 [25]

Figura 32 - Nexys 4 Artix-7 FPGA Trainer Board

5.3.5.1. Comportamiento de un oscilador líder

Antes de simular toda la red, interesaría ver cómo variaría el potencial de una neurona aislada, para comprobar que todas las unidades que compone a un oscilador realizan su función correctamente.

En la Figura 33, se simula un oscilador que está en la esquina arriba-izquierda de la imagen. Se ponen a cero el valor de pixel de los vecinos de arriba y de izquierda, ya que no existen. Aunque las diferencias en el valor de pixel de las 4 entradas están todas dentro del umbral, la fuerza de unión resultante da 1010 (solo hay unión con el vecino de abajo y de derecha) porque los parámetros {has_up, has_left = false} fuerzan las posiciones inexistentes a cero. Como el número de vecinos conectados es 2, esta neurona será líder y tomará un estímulo de entrada igual a 1,25.

Se puede ver que, si se mantiene la imagen de entrada sin cambiar, los valores de fuerza de unión y el estímulo de entrada se mantienen siempre constante. El valor del ruido cambia a cada ciclo de reloj y es de un valor pequeño en comparación con el rango de valores del potencial.

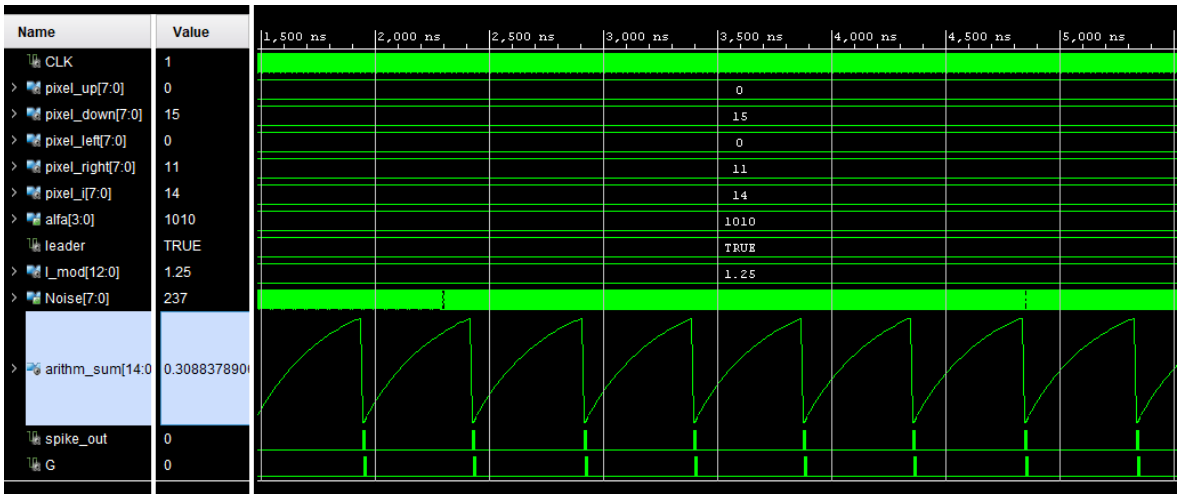


Figura 33 - Comportamiento de un oscilador líder sin estímulo recibido de los vecinos. Simulación en Vivado.

También, se puede apreciar claramente que cuando el potencial llega al valor máximo, se generará un impulso y el potencial se resetea al valor inicial, nulo. Aquí, como se simula solo un oscilador, no se aprecia el efecto de la señal inhibidora sobre la evolución del potencial, pero se puede ver que se pone activa al recibir un potencial de acción.

En las simulaciones, los impulsos de salida es una señal cuadrada perfecta, que en práctica generalmente no alcanza a serlo. Para ver la forma de estos impulsos en práctica, se implementa un solo oscilador en el dispositivo FPGA llevando la salida (Spike_out o impulso de salida) al puerto PMOD y conectado con un osciloscopio.

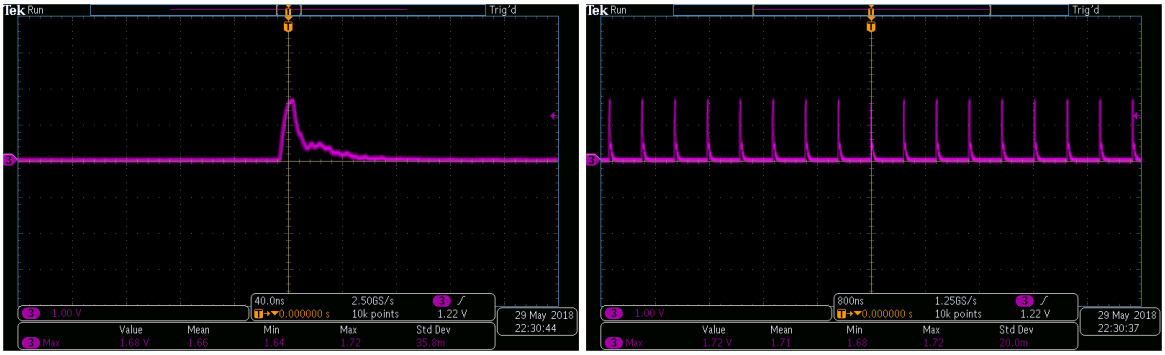


Figura 34 - Impulso de salida de un oscilador leader. Sin estímulo recibido del vecindario. (Izq.: escala horizontal a 40 ns/DIV; Derecha: escala horizontal a 800 ns/DIV)

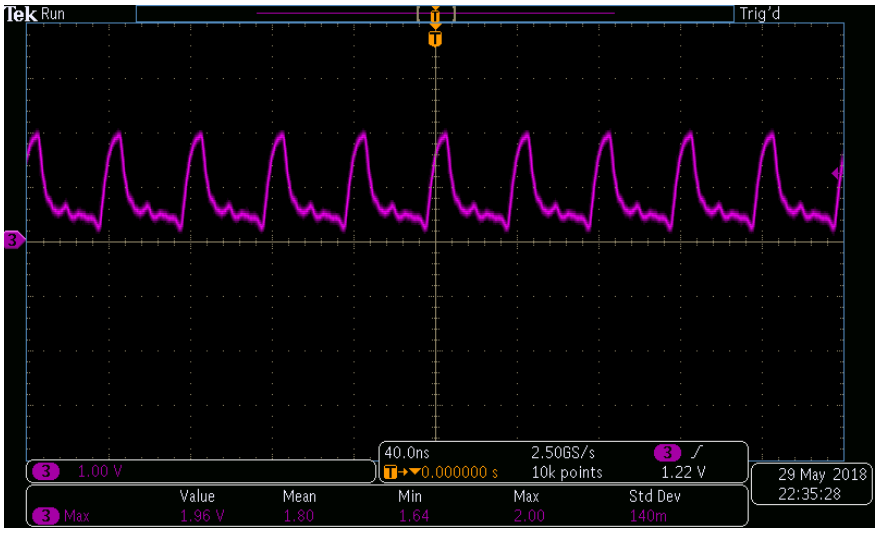


Figura 35 - Impulso de un oscilador leader cuando recibe estímulo de un vecino conectado

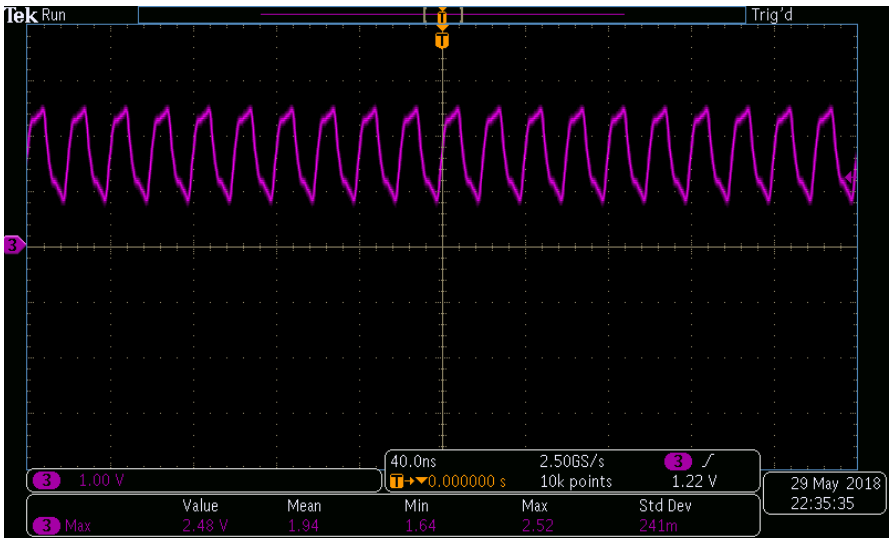


Figura 36 - Impulso de un oscilador leader cuando recibe estímulo de dos vecinos conectados

Tomando un oscilador con 4 vecinos, pero solo dos de ellos conectados, se puede ver el efecto del impulso de los vecinos sobre él. En las Figura 34, Figura 35 y Figura 36 se puede observar que, sin estímulo del vecindario, el oscilador vuelve al potencial de reposo (0 mV) después del impulso; y en segundo lugar, cuanto mayor es el número de impulsos recibidos, mayor será la frecuencia de oscilación y mayor es el potencial de "offset", ya que el oscilador recibe los impulsos del vecindario justo después de ser reseteado.

5.3.5.2. Comportamiento de osciladores no líderes

A diferencia de los líderes, un oscilador no leader con al menos un vecino conectado no tendrá potencial suficiente como para generar un impulso sin recibir estímulo de dichos vecinos. Tal y como se puede comprobar en la Figura 37 y en la Figura 38, el potencial del oscilador se mantiene más o menos constante alrededor de un valor próximo a 1, pero siempre menor que 1. Por lo que nunca producirá un impulso, a menos que recibe un impulso del vecindario, como en la Figura 39.

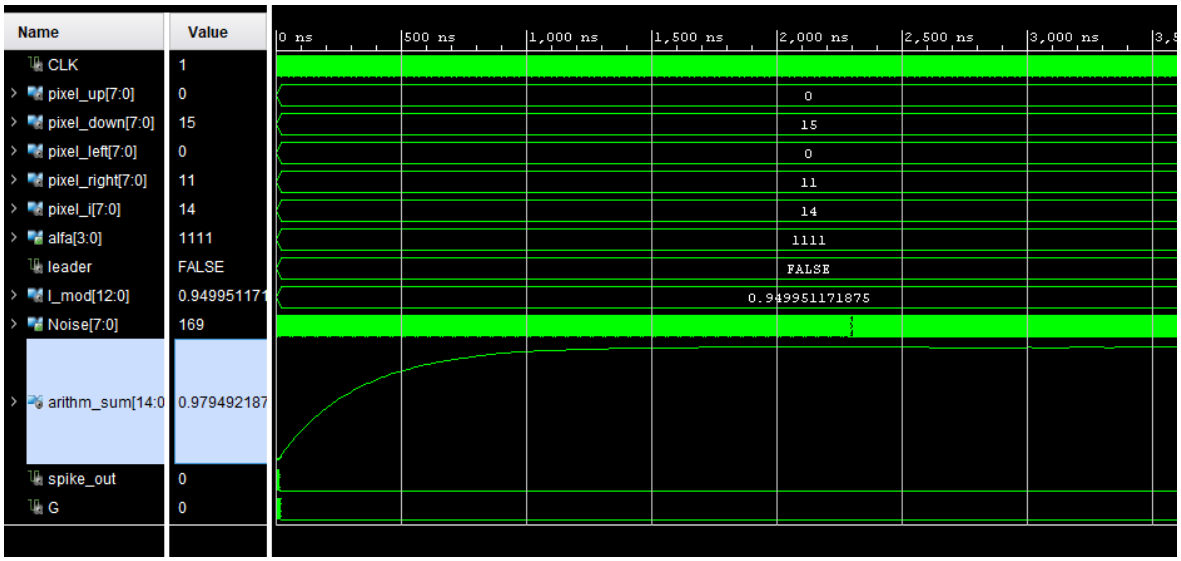


Figura 37 - Oscilador no líder sin recibir estímulo del vecindario. Simulación en Vivado.

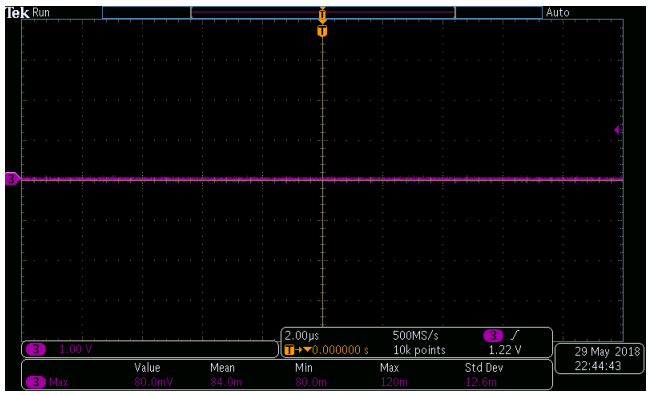


Figura 38 - Estado de impulso de un oscilador no leader sin recibir estímulos de vecindario.

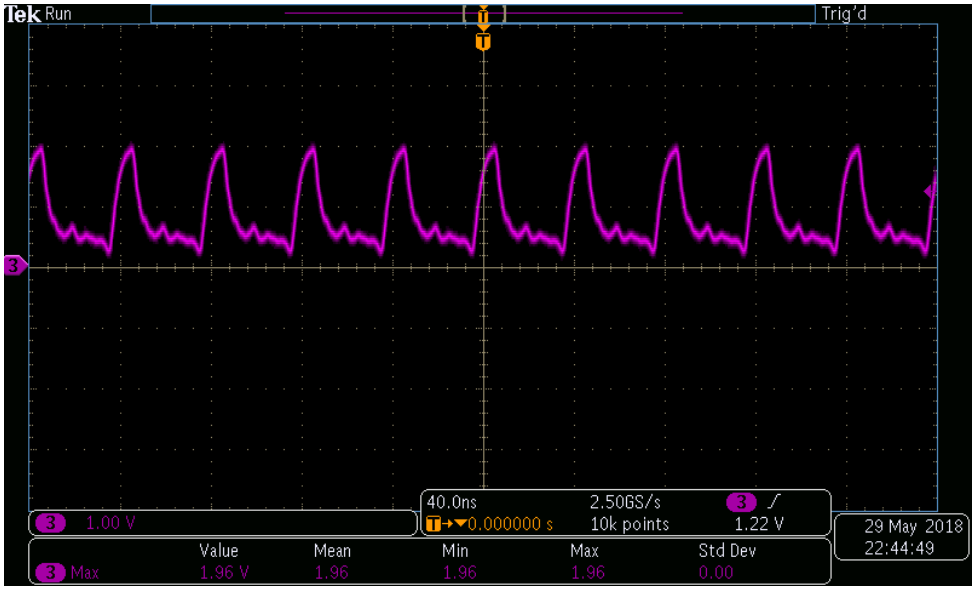


Figura 39 - Estado de impulso de un oscilador no leader cuando recibe impulso de un vecino conectado

Cabe mencionar que el puerto PMOD proporciona señales de salida con una tensión de 3,3 V. Sin embargo, en todas las capturas realizadas con el osciloscopio se observa que los impulsos tienen una amplitud muy inferior a 3,3 V. Ello es debido a que los impulsos tienen una duración muy corta, de 10 nanosegundos, y el circuito de salida del PMOD contiene condensadores. La señal del impulso no tiene tiempo suficiente para llegar a la tensión asignada (3,3 V).

Más adelante se utiliza el osciloscopio Rigol de entradas mixtas, con ancho de banda 100 MHz. Sin embargo, para visualizar el comportamiento de un oscilador (señal analógica) se utiliza otro osciloscopio, Tektronix MO3034, con ancho de banda 350 MHz, debido a que un osciloscopio de 100 MHz no mediría bien una señal de 100 MHz (los impulsos tienen una duración de tan solo 10 ns).

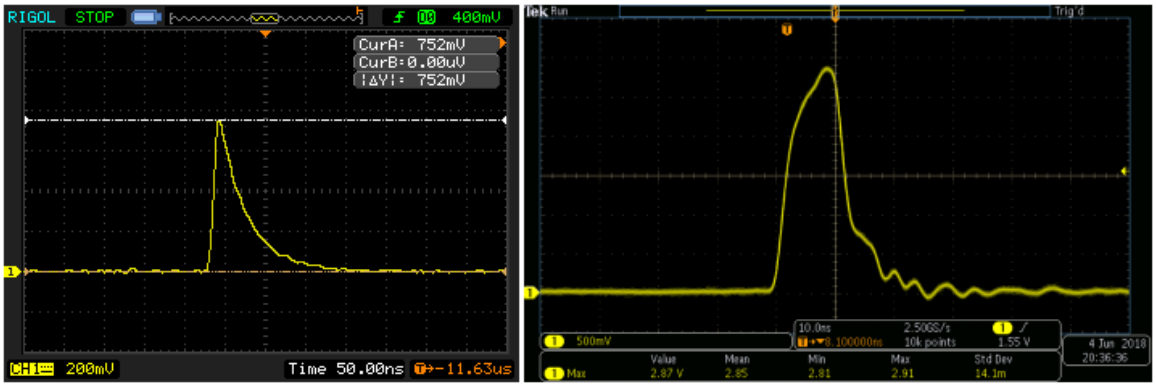


Figura 40 - Captura de una misma señal en Rigol DS1102D (izquierda) y en Tektronix MDO3034 (derecha)

5.3.5.3. Interacción entre osciladores

Viendo ya las diferencias de un líder y uno que no es, sería interesante ver cómo los lideres lideran el disparo de sus vecinos y, por consiguiente, de todos los osciladores de la figura.

La Figura 42 y la Figura 43 muestran la evolución del potencial de tres osciladores que pertenecen a la misma figura, con una distribución en la red como la indicada en la Figura 41, siendo el oscilador 53 un líder.

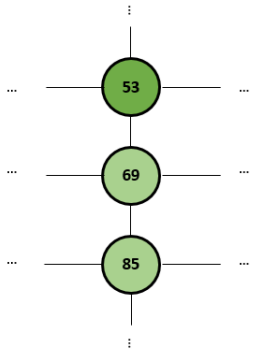


Figura 41 - Distribución de los osciladores del ejemplo de la Figura 42 en la red

Se puede apreciar claramente en la Figura 42 y en la Figura 43 que cuando el líder dispara, el oscilador (#69) que está conectado con él recibe su impulso y dispara también en el siguiente ciclo de reloj. Este impulso que recibe se refleja en un pico en la evolución del potencial. El oscilador que está conectado con éste recibe su impulso y dispara también, pero un ciclo de reloj más tarde. Así pues, el impulso del líder se propaga a todos los osciladores que pertenecen al mismo grupo, con los coeficientes bien ajustados.

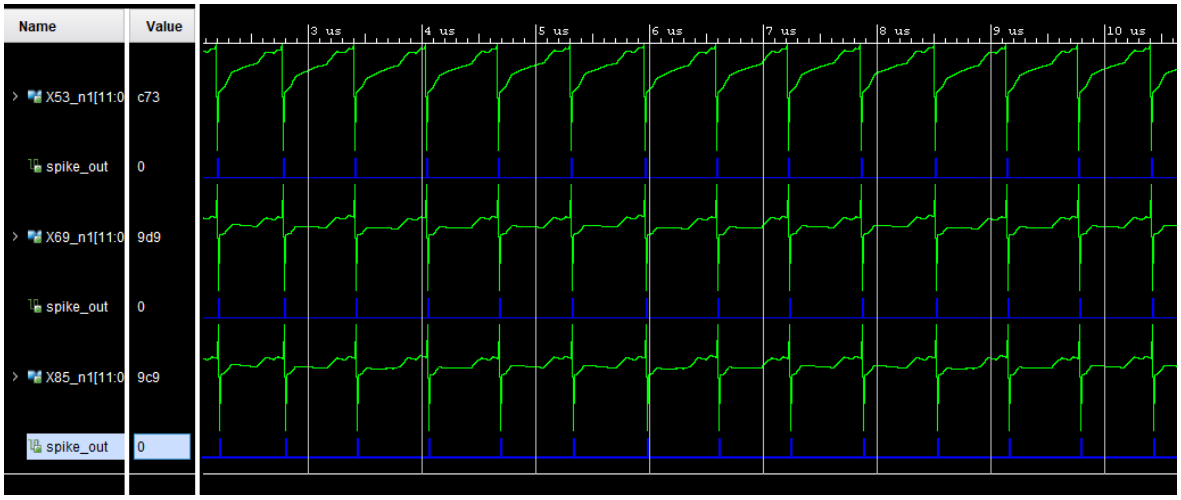


Figura 42 - Un líder y dos osciladores arrastrados

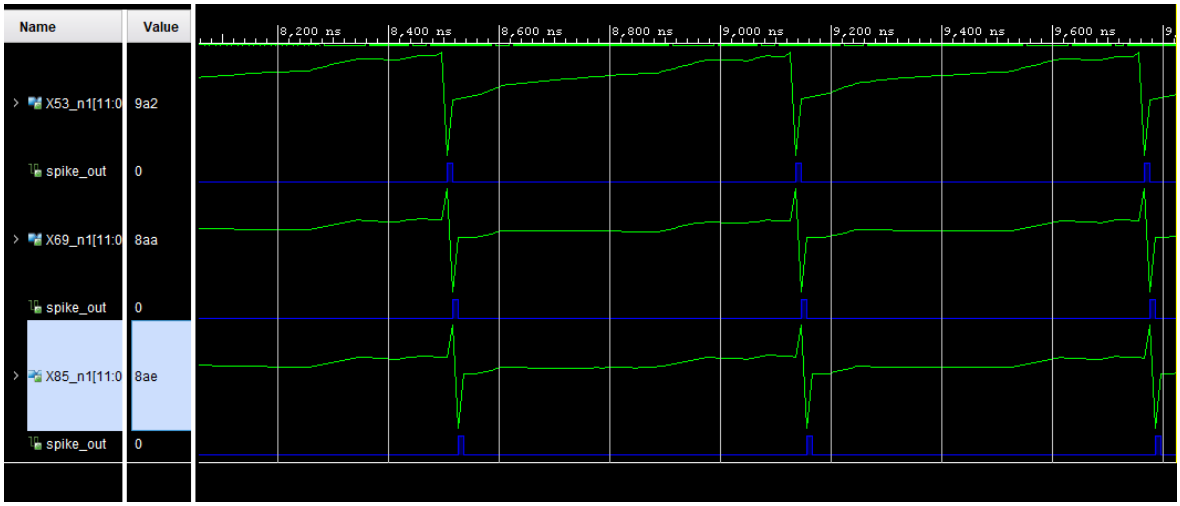


Figura 43 - Imagen aumentada de la Figura 42

Se observa también que el oscilador adquiere un potencial grande justo después del potencial de reposo, debido a que recibieron impulsos de otros osciladores vecinos conectados que aquí no se ha mostrado por limitación de espacio.

Este efecto de propagación de impulsos se puede apreciar tanto en las simulaciones realizadas con red de 192 neuronas (Figura 46) como en las capturas (Figura 44 y Figura 45) realizadas en el osciloscopio de los osciladores de la Figura 41.

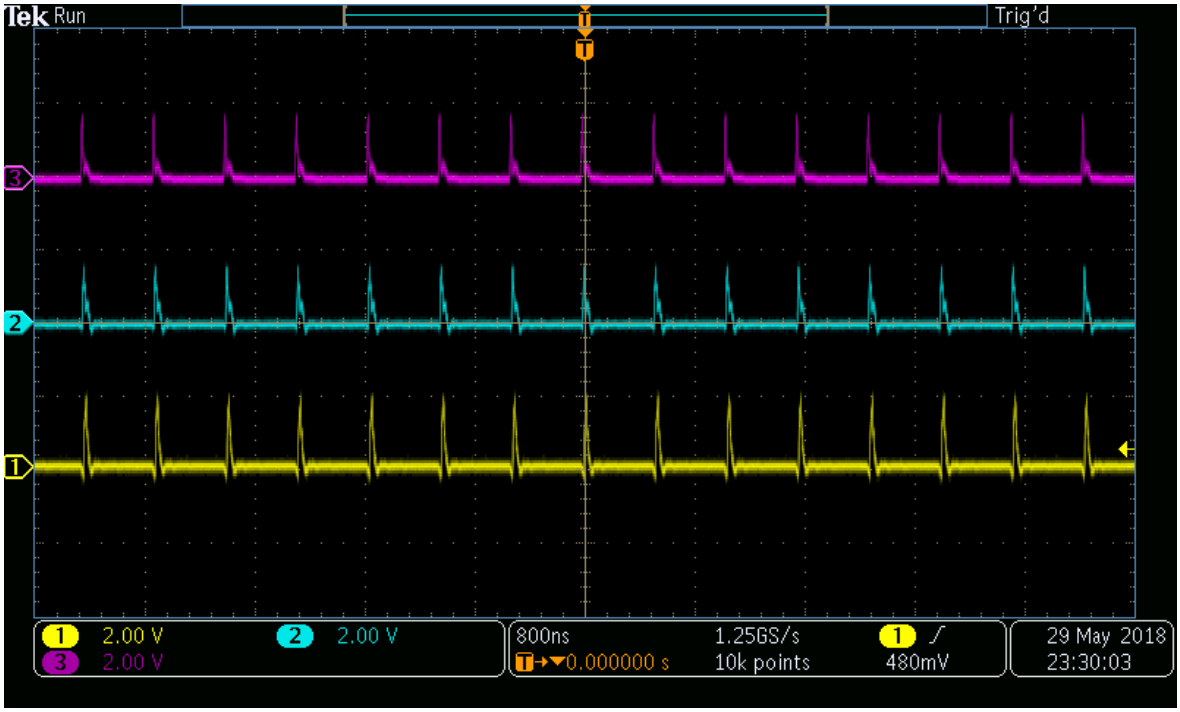


Figura 44 - Un leader (en lila) y dos osciladores arrastrados (el azul por el leader y el amarillo por el azul)

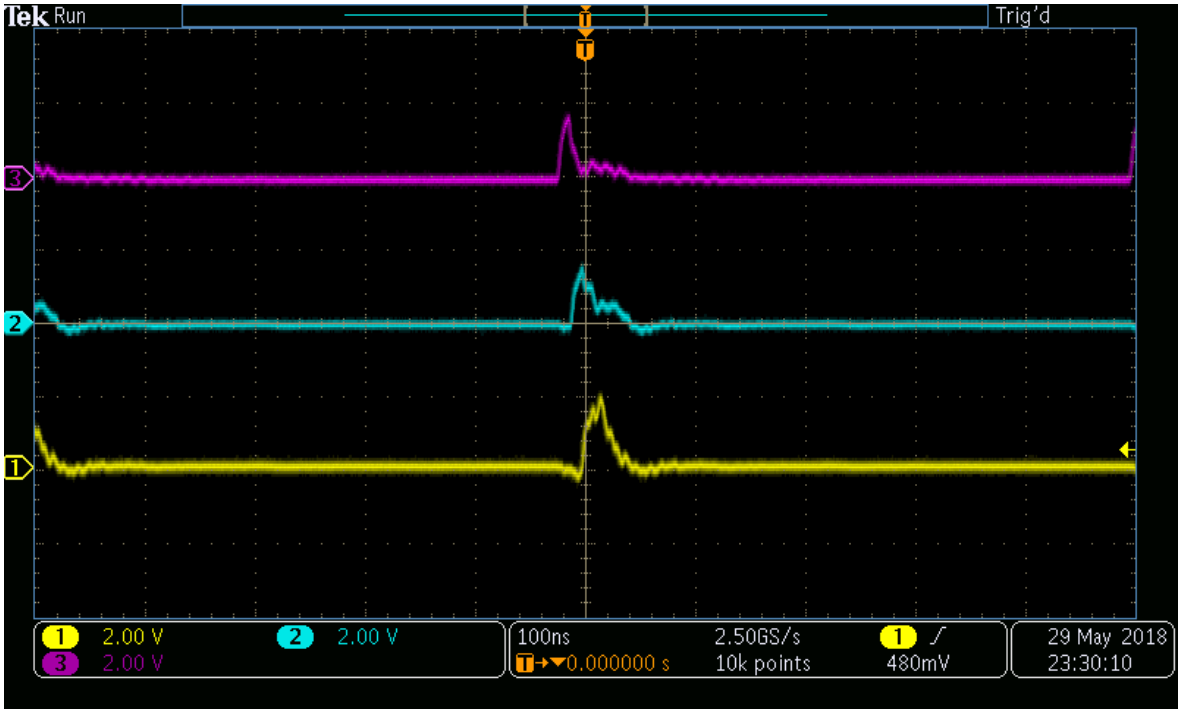


Figura 45 - Figura 44 con escala horizontal a 100 ns/DIV

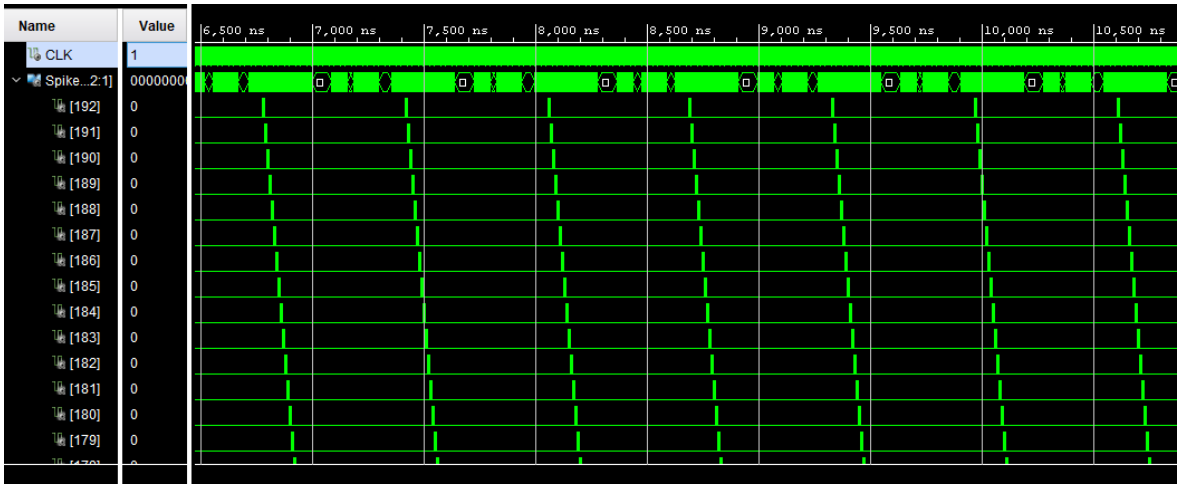


Figura 46 - propagación de los impulsos en un grupo de osciladores

El potencial de los no líderes crece rápidamente al recibir un impulso del vecino, líder o no líder. El coeficiente de inhibición, G_value, por lo tanto, no puede ser de un valor muy grande como para inhibir el disparo de estos osciladores, pero sí suficiente como para inhibir los del grupo diferente. Se ha encontrado empíricamente el valor de G_value/32, comprendido entre [40, 55] en la representación de punto fijo de 12 bits.

Por el mismo efecto de propagación de impulsos, un solo líder no tendrá fuerza suficiente como para poder inducir estos disparos encadenados. De ahí que uno de los retos de este proyecto está en encontrar un criterio bueno de líder y los coeficientes G_value y w_value. Estos dos coeficientes son característicos del ancho de intervalo 1/h, por lo que es crucial fijar el ancho de intervalo desde el inicio del diseño.

5.3.5.4. Otros fenómenos en la red

El fenómeno de propagación de impulsos se produce una vez que la red se ha estabilizado. Antes de este periodo de estabilización, la red pasa por un periodo de transición cuya duración depende de la imagen de entrada.

En el periodo de transición, los impulsos dentro de un mismo grupo pueden no estar sincronizados o tener un comportamiento inusual. El mecanismo de sincronización no consigue inmediatamente la sincronización de los grupos, el tiempo que tarda en hacerlo es el periodo de transición.

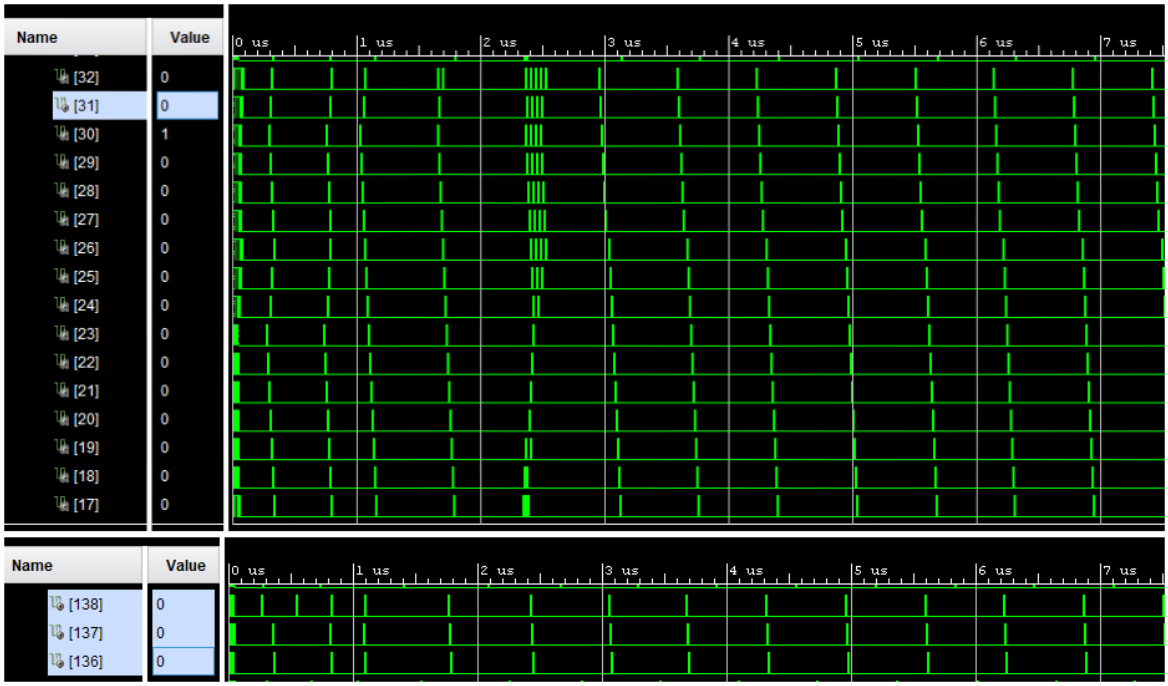


Figura 47 – diferentes periodos de transición

Por último, cabe mencionar que utilizando un FPGA de la familia Artix7 con modelo XC7A100T-CSG324 se puede llegar a implementar una LEGION de 550 osciladores aproximadamente. Este cálculo es aproximado, basándose en los informes de utilización (Utilization Report) de Vivado.

Tabla 1 - Uso de recursos de la red dependiendo de su tamaño

Tamaño de red	*LUT (63400 disponible)*	*FF (126800 disponible)*
8x6 (48)	5342 (8,43%)	1776 (1,4%)
16x12 (192)	21064 (33,22%)	7104 (5,60%)
25x20 (500)	54379 (85,77%)	18492 (14,58%)
25x22 (550)	59713 (94,18%)	20334 (16,04%)
25x24 (600)	65073 (102,64%)	22184 (17,50%)
30x20 (600)	65148 (102,76%)	22184 (17,5%)
33x30 (990)	106842 (168,52%)	36606 (28,87)

6. Implementación de LEGION en FPGA

Como la red diseñada está en la capa oculta, se necesita crear una capa de entrada y otra de salida para tener control la entrada y poder visualizar los resultados.

6.1. Entradas

Para poder comprobar la funcionalidad del sistema electrónico diseñado con fin de emular redes neuronales, en este caso LEGION, se utiliza una serie de imágenes prediseñadas para ver el comportamiento de la red.

6.1.1. Imágenes de entrada

Las imágenes prediseñadas son un vector de longitud igual a la cantidad de neuronas en la red con valores en un rango de [0, 255] para facilitar la conversión al vector de 8 bits posteriormente.

Aunque se puede implementar una red de hasta 550 neuronas, por el coste de diseño se han hecho todas las simulaciones e implementaciones posteriores con una red de 16x12. Teniendo en cuenta que, en Vivado, una red de 16x12 ya tarda alrededor de 35 minutos en poder generar un Bitstream (tiempo incluyendo Synthesis + Implementation), trabajando con un portátil con procesador Intel i5.

Un ejemplo de imagen prediseñada sería:

```
img1_aux <=
(250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
 250, 111, 111, 111, 111, 111, 111, 250, 250, 250, 250, 250, 250, 250, 250, 250,
 250, 111, 111, 111, 111, 111, 111, 250, 250, 250, 250, 250, 250, 250, 250, 250,
 250, 111, 111, 111, 111, 111, 111, 250, 250, 250, 50, 50, 50, 50, 250, 250,
 250, 111, 111, 111, 111, 111, 111, 250, 250, 250, 50, 50, 50, 50, 250, 250,
 250, 111, 111, 111, 111, 111, 111, 250, 250, 250, 50, 50, 50, 50, 250, 250,
 250, 111, 111, 111, 111, 111, 111, 250, 250, 250, 50, 50, 50, 50, 250, 250,
 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250);
```

Este vector representaría la siguiente imagen:

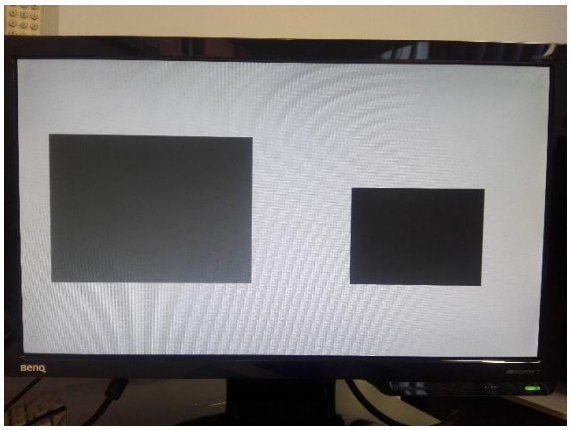


Figura 48 - Ejemplo de una imagen prediseñada

6.1.2. Selección de la imagen

Para no volver a sintetizar todo el código y esperar media hora para ver los resultados (este es el tiempo de implementación del algoritmo en el dispositivo FPGA, no el tiempo de procesamiento de la red), se guardan varias imágenes prediseñadas en la memoria. Para elegir con qué imagen trabajar, incluyendo la imagen de reset, se utilizarán unos interruptores.

De esta forma, la capa de entrada tiene la función de convertir la imagen seleccionada en un array de vectores y lo transmite a la capa oculta, la red.

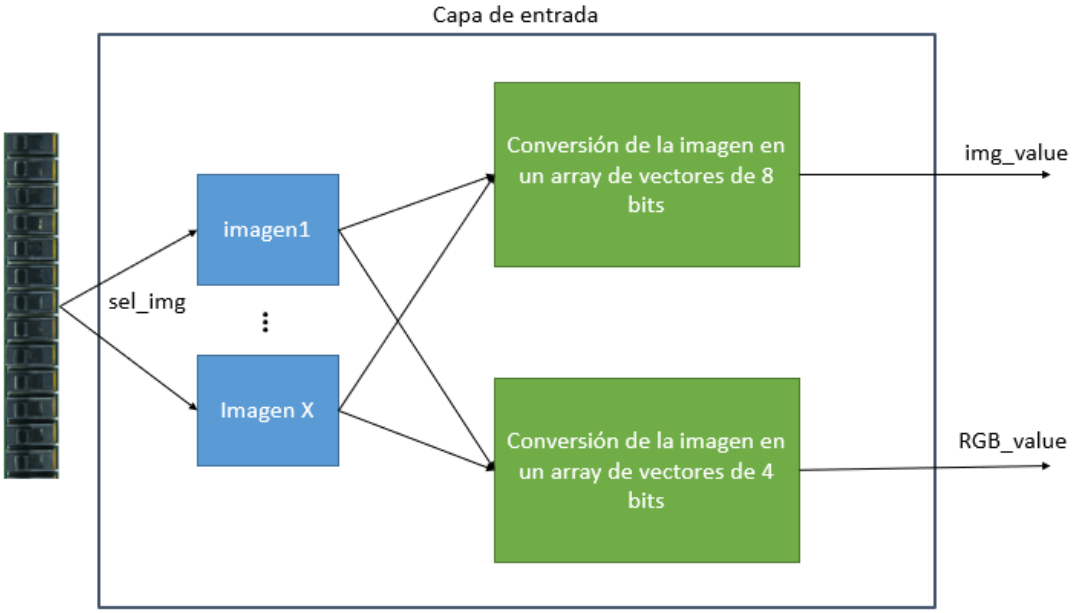


Figura 49 - Arquitectura general de la capa de entrada

6.2. Salidas

Como la LEGION localiza figuras en una imagen, sería lógico que la salida de la red fuese un monitor donde poder exponer las figuras segmentadas. Además del monitor, se utiliza un osciloscopio con 16 entradas digitales para visualizar los impulsos de algunos osciladores en la red, y un display de 7 segmentos para mostrar el estado de visualización.

Sin embargo, no es posible llevar directamente la salida de la red, que son los impulsos de cada uno de los osciladores (Spikes_out), al monitor. Por un lado, la red en funcionamiento produce impulsos a una frecuencia de cientos kHz y la duración de cada impulso es de tan solo 10 nanosegundos, muy superior a la frecuencia que el ojo humano puede apreciar. Por el otro lado, el tamaño de la red no corresponde con la resolución que ofrecen los monitores comerciales.

Se presentará a continuación la resolución de estos dos problemas y los medios que se utilizan para visualizar la salida.

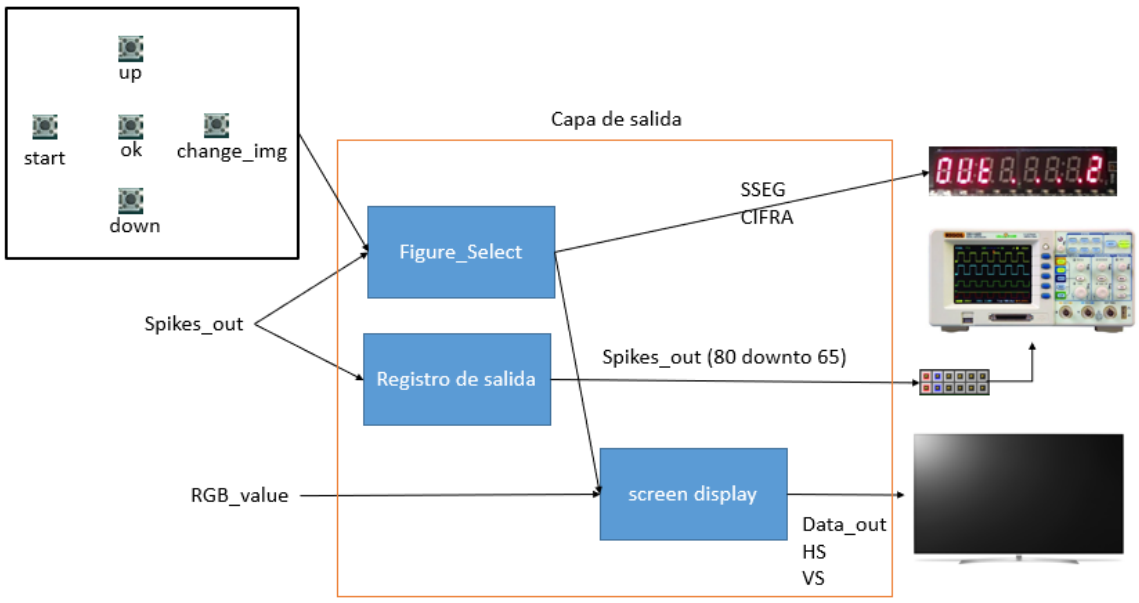


Figura 50 - Arquitectura general de la capa de salida

6.2.1. Selección de patrones encontrados

Para resolver el primer problema mencionado, se ha diseñado un bloque Figure_select.

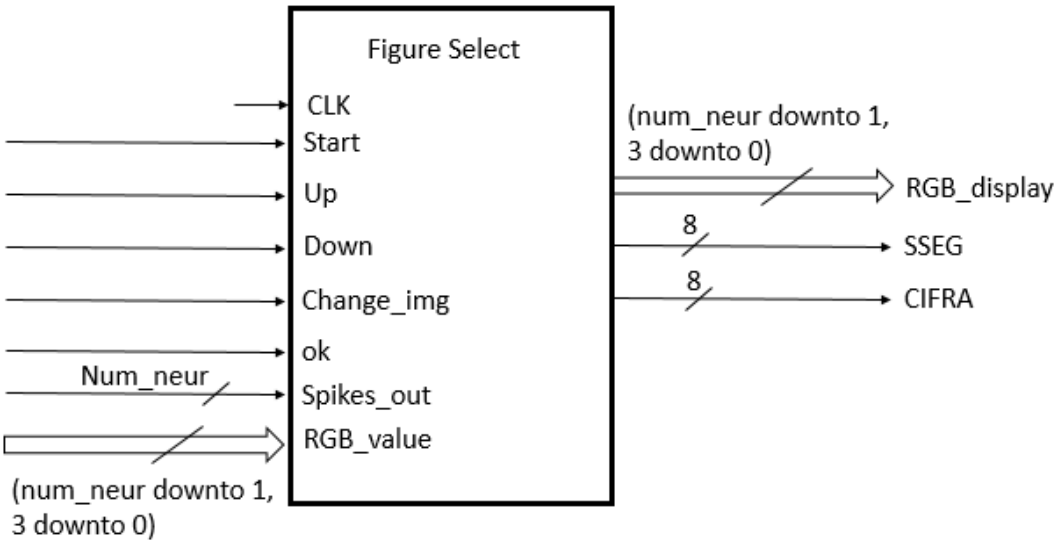


Figura 51 - Arquitectura del bloque Figure_select

La estrategia consiste en buscar las figuras reconocidas por la red y guardarlas en una memoria. Este proceso se lleva a cabo mediante los siguientes pasos:

1- Asignar a un botón la función de empezar a grabar las figuras encontradas. Con esto se garantiza que la red ya se ha estabilizado al empezar a buscar las figuras, ya que el tiempo de estabilización es menor que 10 µs y al apretar un botón se tardan unos milisegundos. Este botón se llamará Start.
2- No empezar a grabar cuando los osciladores estén disparando, porque es muy posible que se pierda parte de la información, y la figura grabada no está completa.
3- Diseñar un mecanismo para poder detectar las figuras y grabarlas
4- Establecer un criterio para terminar el proceso y estar preparado para la siguiente imagen a analizar
5- Transformar la figura seleccionada (en B/N) o la imagen original (en escala de grises) en una única salida, RGB_display. Esta transformación se explicará en el apartado 6.2.4.4

Inicio de la grabación

Se aprovecha la máquina de estados del botón Start para realizar esta función añadiendo una condición: no enviar la señal Start_aux (señal que indica el inicio de grabación) inmediatamente después de dejar de apretar el botón, sino esperando a que ningún oscilador en la red esté disparando. Este último se consigue con mirar si el vector Spikes_out, donde se recogen los impulsos de todos los osciladores de la red, es nulo o no.

Algoritmo de detección de figuras y fin del proceso

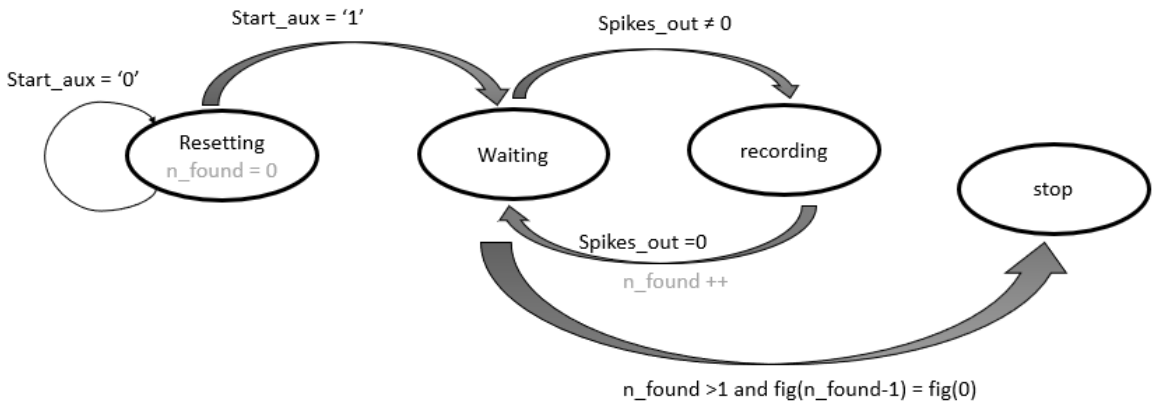


Figura 52 - Máquina de estado finito (FSM) para detectar presencia de una figura

Este algoritmo es básicamente una máquina de estado finito (FSM) de cuatro estados. Al recibir orden de empezar a grabar (Start_aux), se pasa al estado de espera (waiting) donde se espera a que hayan impulsos en la red. Entre un grupo de impulsos y el otro siempre hay una margen en donde no hay disparos en la red. Esta es la característica más importante que define la LEGION, en donde cada grupo de oscilaciones representa una figura.

Al detectar un impulso en la red, se pasa al estado de grabación. Se vuelve al estado de espera si la red vuelve al estado de reposo. Y solo en este cambio de estado se incrementa en uno la variable n_found, que es el número de figuras encontradas.

El proceso de grabación se acaba cuando ya se ha encontrado más de una figura y la última figura que se ha encontrado es idéntica a la primera encontrada, con un simple comparador.

Estando en cualquier estado, la FSM vuelve al estado resetting si recibe una señal activa de reset, que borra la memoria de figuras encontradas y resetea n_found, preparándose para una nueva búsqueda. Esta señal es manual, con un botón que se denominará change_img.

Algoritmo de grabación de figuras

Este proceso depende de la variable n_found. Para contener las figuras encontradas se define un array de vectores. Estos vectores tienen la misma longitud que el tamaño de la red y cada bit se corresponde con una posición del pixel en la imagen en donde el valor '1' simboliza que el pixel correspondiente forma parte de una figura. Se utiliza n_found como índice del array para localizar a cualquier figura encontrada.

La grabación empieza con el array reseteado a vectores nulos. Y consiste en poner a '1' solo a aquellas posiciones en el vector correspondiente al índice de valor n_found, en donde se ha producido un impulso, dejando a las demás posiciones intactas ('0').

Parte del código de este algoritmo es:

```
if (rec_state = waiting) or (rec_state = recording) then
    for i in 1 to num_neur loop
        if (spikes_out(i) = '1') then
            fig(to_integer(n_found))(i) <= spikes_out(i);
        end if;
    end loop;
end if;
```

Con la señal de reset, este array se resetea.

6.2.2. Puerto PMOD

Aunque no es posible visualizar los impulsos de salida de todos los osciladores de la red en un osciloscopio, se puede hacerlo con una línea de osciladores mediante un osciloscopio digital de 16 entradas digitales. Solo hace falta conectar la salida de, por ejemplo, los osciladores de la 5ª línea en la imagen (los osciladores de 65 a 80), con dos de los puertos PMOD que proporciona el FPGA Artix7.

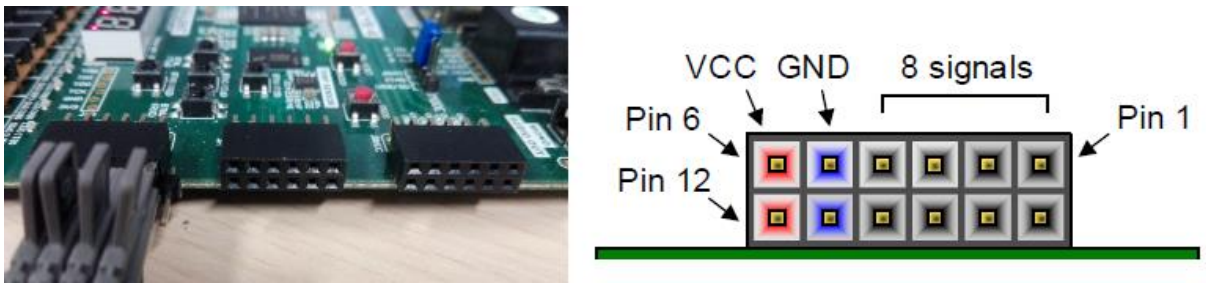


Figura 53- imagen de izquierda: Puerto PMOD en Artix7. Imagen de derecha [26]*: conexión del puerto PMOD*

6.2.3. Display de 7 segmentos

Se utiliza un display de 7 segmentos en combinación con tres botones más (llamados up, down y ok para la selección) para visualizar el número de figuras encontradas y seleccionar qué figura se desea visualizar en el monitor.

Para realizar estas funciones también se ha diseñado una FSM:

a) En ausencia de instrucciones o tras presionar change_img, se expone una serie de puntos

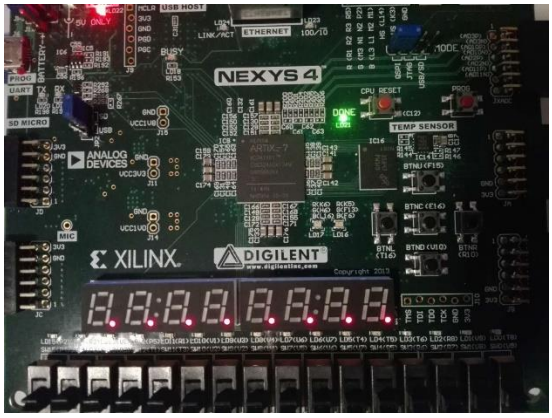


Figura 54 - Estado: esperando instrucciones

b) Al presionar el botón start se expone la cantidad de figuras encontradas en la imagen

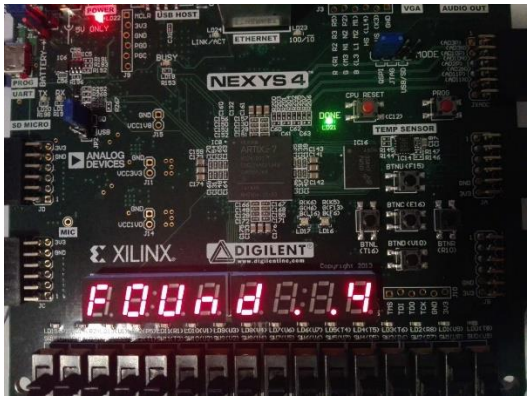


Figura 55 - Estado: indicando el número de figuras encontradas

c) Al presionar los botones up y down se selecciona la figura que se desea ver

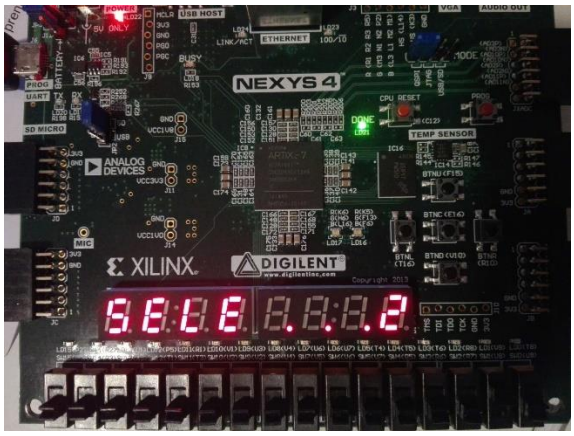


Figura 56 - Estado: selección de la figura a visualizar

d) Al presionar el botón ok, se enviará la figura seleccionada al bloque que prepara este vector para ser visualizado en el monitor

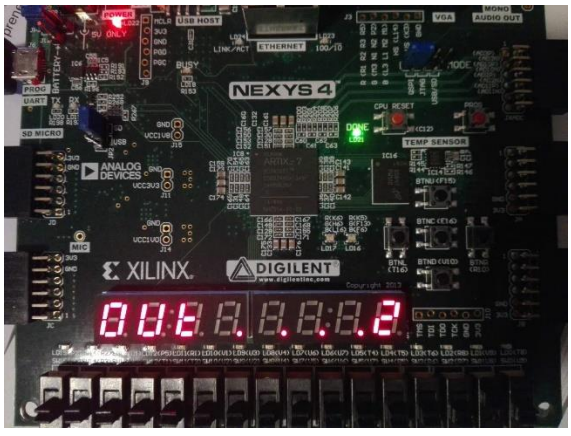


Figura 57 - Estado: visualizando la figura seleccionada

6.2.4. Puerto VGA

Una vez que se ha seleccionado la figura a visualizar, se pondrá en la pantalla una imagen en donde el color negro simboliza la presencia de la figura y el color blanco, el fondo.

Para ello se explicará primero el modo de funcionamiento de pantallas VGA basadas en tubos de rayos catódicos (CRT).

6.2.4.1. Funcionamiento de monitores de tubos de rayos catódicos

El funcionamiento de un monitor CRT [26] consiste en el choque de los haces de electrones, producidos y confinados por el cañón de electrones, contra la pantalla recubierta de material fluorescente, como el fósforo, para producir una luz controlada en intensidad en un punto concreto de la pantalla. Los monitores CRT de color utilizan 3 haces de electrones (uno para el rojo, uno para el verde y el otro para el azul), producidos por 3 cañones de electrones, y 3 capas de fósforo, una por cada color básico. Combinando las intensidades de los haces de electrones de los tres colores básicos se produce el color deseado. Para poder enviar estos haces de electrones al punto deseado en la pantalla se colocan unas bobinas de deflexión magnética después de los cañones de electrones. Estas bobinas producen un campo magnético suficientemente grande como para poder desviar los haces de electrones a cualquier punto deseado en la pantalla.

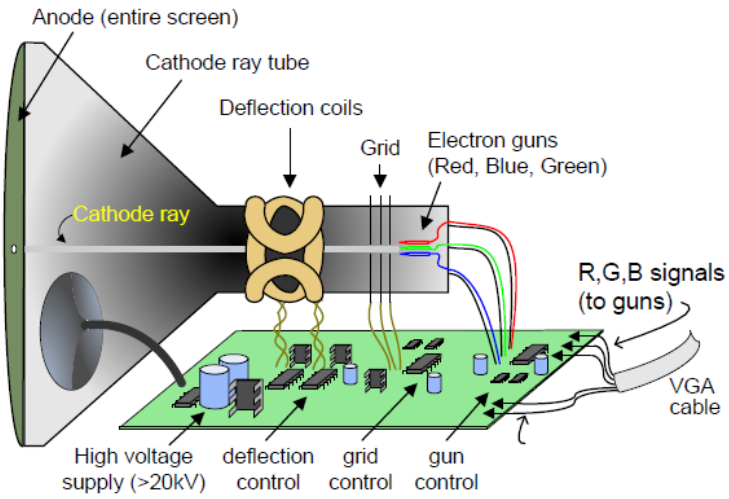


Figura 58 - Partes de un monitor VGA basado en CRT [26]

Esta desviación de haces no se hace de forma aleatoria, sino que sigue una secuencia: de izquierda a derecha y de arriba abajo, recorriendo toda la pantalla. De esta forma, en cada instante, solo hay un pixel iluminando, pero a una frecuencia que el ojo humano no es capaz de percibir y se vería que todos los pixeles en la pantalla se iluminan a la vez.

6.2.4.2. VGA system timing

Contrario a la intuición, cuando se acaba de iluminar la primera línea de pixeles en la pantalla, no se empieza inmediatamente a iluminar la segunda línea, ya que se necesitaría una diferencia de tensión muy grande en las bobinas para cambiar la posición de un extremo al otro. La solución fue en apagar las bobinas de deflexión durante cierto tiempo y redirigir la dirección para poder empezar en el primer pixel de la segunda línea. Durante este tiempo de cambio de línea, no se visualiza ninguna información. Por la misma razón, cuando se acaba de proyectar todos los pixeles de la pantalla, se necesitará un tiempo (mayor que el de cambio de línea al ser mayor distancia que recorrer) para poder proyectar la siguiente imagen.

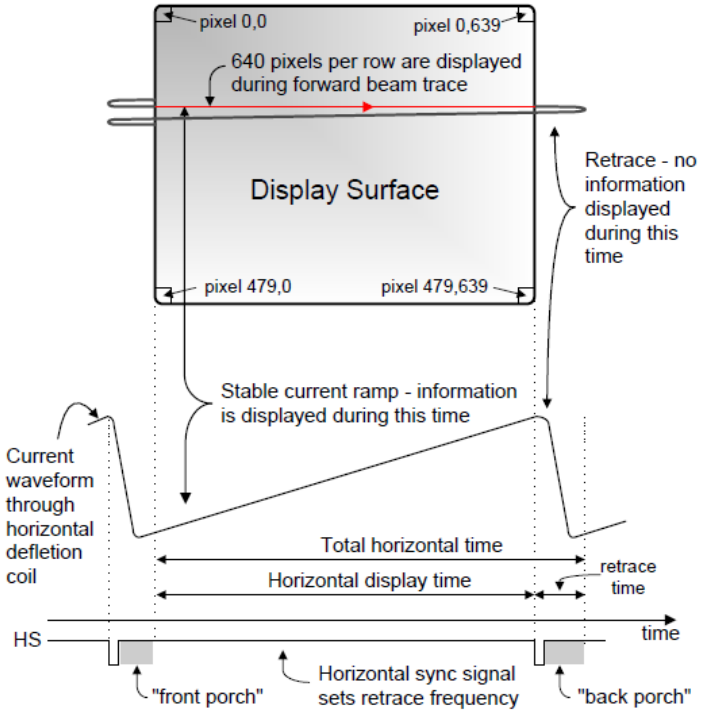
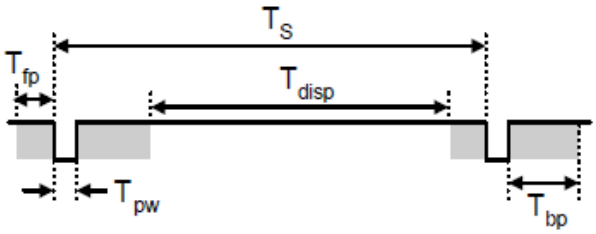


Figura 59 - Sincronización horizontal VGA [26]

El estándar o la norma VGA nace de esta necesidad de sincronización de tiempos de visualización y de recuperación o transición. Con la norma VGA se estandariza la frecuencia de refresco, los tiempos de visualización, etc. VESA (www.vesa.org) es la organización que especifica y vende la sincronización del sistema VGA (VGA system timing) con derechos del autor.

La norma VGA no se aplica solo a monitores CRT, puede ser aplicada también, entre otras, a monitores LCD [26] y de LED. El monitor que se utiliza en este trabajo es de LED, un BenQ G922HDL.

Las pantallas VGA modernas pueden acomodar diferentes resoluciones, y un circuito controlador VGA determina la resolución produciendo señales de sincronización (HS y VS) para controlar los patrones de trama. Los tiempos de las señales (signal timings) dependen de la resolución escogida, típicamente de entre 240 a 1200 filas y de entre 320 a 1600 columnas. Para este trabajo, se utiliza una resolución de 640x480 con un reloj de 25 MHz y una frecuencia de refresco de 60 Hz, cuyos tiempos de sincronización correspondientes se recogen en la Figura 60.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figura 60 - Tiempos de sincronización (signal timings) para una resolución de 640x480 utilizando un reloj de 25 MHz y una frecuencia de refresco de 60 Hz [26]

Para una sincronización correcta, además de cumplir con los tiempos de señal (signal timings), se ha de generar dos señales de sincronización, HS y VS: HS indica al monitor el cambio de línea y VS, cambio de imagen.

Resumiendo, para poder visualizar una línea en la pantalla de 640 pixeles por 480 filas, se ha de enviar primero la señal HS durante 96 ciclos de reloj, antes de enviar señales de información se espera 48 ciclos de reloj, y se espera otros 16 ciclos de reloj antes de empezar el siguiente ciclo o línea. De la misma forma, se envía la señal VS durante 2 líneas de tiempo (2x800 ciclos de reloj), y se espera 29 y 10 ciclos de reloj antes y después de enviar la señal de información.

Si se traduce estos signal timing y timing signals (HS, VS) en una imagen, se vería como:

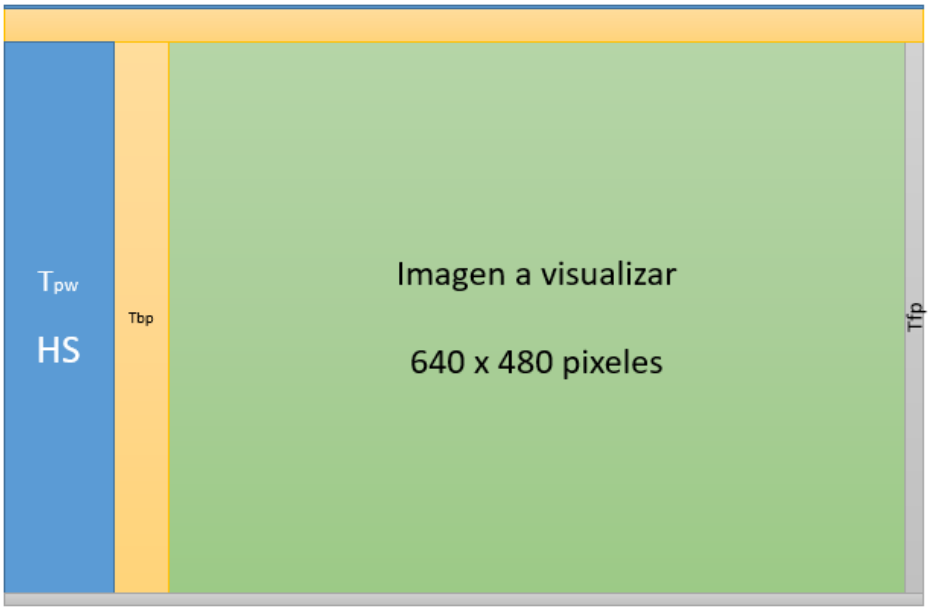


Figura 61 - Imagen que se envía al monitor

Los rectángulos verticales simbolizan sincronización horizontal y los rectángulos horizontales, sincronización vertical. Durante los tiempos Pulse Width (T_{PW}, en azul) se envían las señales HS y VS correspondientes, y durante los tiempos Back Porch (T_{bp}, en amarillo) y Front Porch (T_{fp}, en gris) no se envía información, o enviando ceros.

Así pues, solo se envían datos de la imagen al monitor cuando la señal por el puerto VGA está en la zona de visualización (T_{disp}, en verde).

6.2.4.3. Adaptación del tamaño de la imagen en resolución completa de la pantalla

Otro de los problemas mencionados en 6.2 es que la imagen de salida es de 16 x 12 pixeles, muy menor en comparación con resolución de la pantalla. El monitor que se utiliza puede llegar a tener una resolución de 1366x769 pixeles, pero con sistema VGA se puede adaptar a una resolución de 640x480.

Para adaptar la imagen de salida a la resolución deseada, se puede aumentar la imagen 40 veces tanto en anchura como en longitud. El aumento consiste en replicar cada pixel de la imagen 40 veces en una línea y en 40 líneas en las mismas posiciones.

Para hacer esta réplica, se crean dos vectores *X* y *Y* en donde *X* aumenta en 1 cada 40 pixeles en la zona de visualización (T_{disp}), y el vector *Y* aumenta en 1 cada 40 líneas en la misma zona. Como la enumeración de los pixeles de la imagen no es bidimensional (coordenadas XY) sino unidimensional (de 1 a num_neur), se hace la siguiente conversión:

$$i = X + image_{length} \cdot Y$$

Donde X, Y son las coordenadas del pixel en cuestión, $image_{length}$ es la longitud de la fila de la imagen (en este caso, 16 pixeles), i es la posición del pixel a replicar.

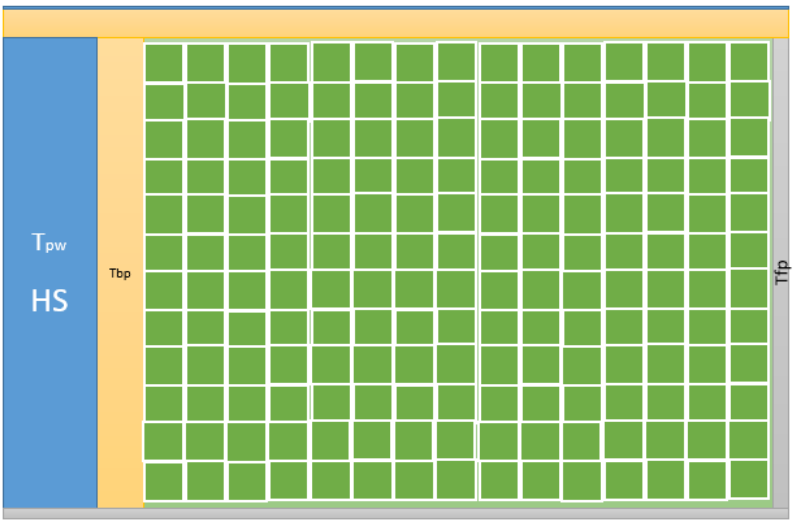


Figura 62 - Adaptación de una imagen de 16x12 a una resolución de 640x480

Ahora, cada pixel de la imagen de salida se visualiza en la pantalla 40 veces mayor.

6.2.4.4. Visualización de la imagen original en pantalla

Y para poder contrastar mejor los resultados obtenidos con la imagen original, se ha establecido una conexión directa entre la capa de entrada y la capa de salida. Cuando el sistema espera por instrucciones, la pantalla siempre expondrá la imagen original.

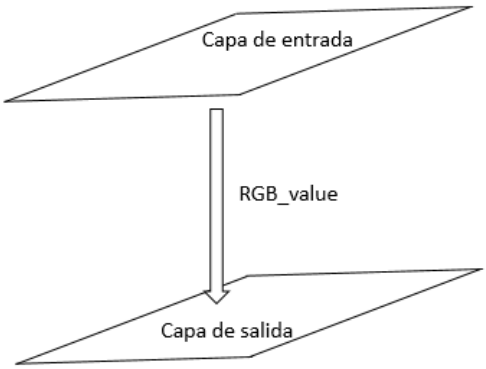


Figura 63- Conexión entre capa de entrada y capa de salida para transmitir los valores de los pixeles de la imagen original en escala de grises

Como se ha explicado anteriormente, la imagen de entrada es un array de vectores de 8 bits. Sin embargo, el Trainer Board Nexys4 DDR, el FPGA de la familia Artix7 que se utiliza en este trabajo, utiliza 14 señales por el puerto VGA: 4 bits por color y 2 señales de sincronización (HS y VS). De esta forma, para visualizar la imagen original en la pantalla, se necesita convertir la escala de grises de 8 bits en color RGB de 12 bits.

El color gris es el resultado de combinar los tres colores básicos (RGB, Red Green Blue) en proporciones iguales. Dependiendo de la intensidad de los colores básicos, se consigue un gris más o menos oscuro. Por ejemplo, “000000000000” es el color negro, “0001 0001 0001” es un gris muy oscuro, “1110 1110 1110” es un gris muy claro y “1111 1111 1111” es el color blanco.

De esta forma, para representar gris en RGB, solamente ha de convertir una escala de grises de 8 bits en una de 4 bits y replicar estos 4 bits tres veces. Un vector de 8 bits representa 256 valores y uno de 4 bits, 16 valores. Por lo tanto, se ha de reducir la resolución original 16 veces.

A nivel de hardware, la reducción se puede conseguir mediante un desplazamiento aritmético hacia la derecha de 4 bits, o simplemente ignorando los cuatro bits menos significativos. Dichos bits tienen el valor máximo de 15, de ahí que el resultado de dividir el vector entre 16 es el mismo que mirar los cuatro bits más significativos del vector.

El código en VHDL de esta conversión es:

```
gray_to_RGB:
for i in 1 to num_neur generate
    im_RGB(i) <= im_value(i)(7) & im_value(i)(6) & im_value(i)(5) &
im_value(i)(4);
end generate;
```

El array resultante estará formado por vectores de 4 bits, los más significativos del vector original.

Por último, el VHDL no permite entradas o salidas definidas como un array de vectores. Para hacerlo posible, se recurre a crear un paquete personal donde se crea un array bidimensional de STD_LOGIC o bit. Por esta razón, aunque en la transmisión entre bloques se utiliza este array bidimensional, se ha de convertir en un array de vectores para su procesamiento o viceversa, convertirlo en un array bidimensional una vez que se ha procesado para enviar los datos al otro bloque. Esta conversión entre arrays se utiliza también para la transmisión de los valores de pixel entre la capa de entrada y la oculta, la LEGION.

Por lo que el código completo de la réplica y la conversión sería:

```
gray_to_RGB:
for i in 1 to num_neur generate
    im_RGB(i) <= im_value(i)(7) & im_value(i)(6) & im_value(i)(5) &
im_value(i)(4);
    pixel_to_pixel:
        for b in 0 to 3 generate
            RGB_value(i,b) <= im_RGB(i)(b);
        end generate pixel_to_pixel;
end generate;
```

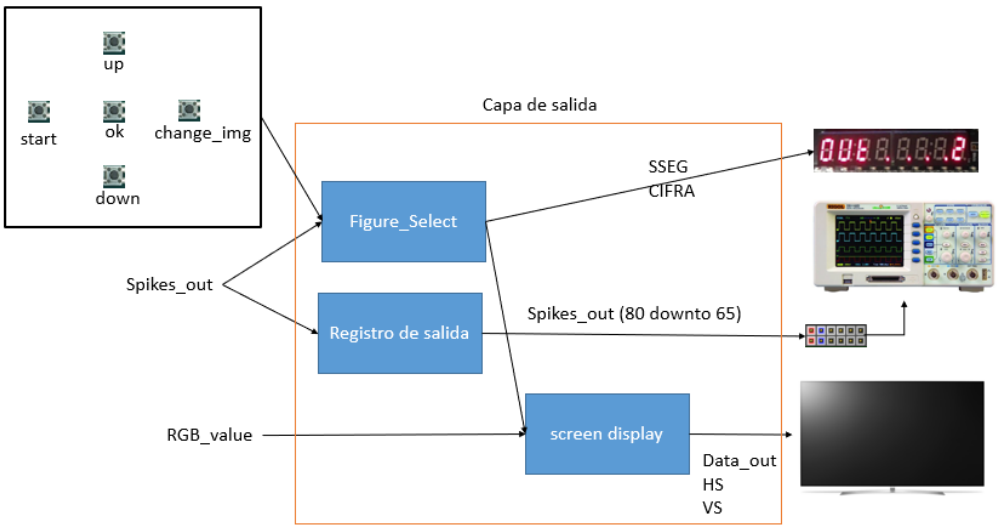


Figura 64 - Arquitectura general de la capa de salida

6.3. Arquitectura de todo el sistema

En el apartado 5 se ha explicado la LEGION y en este apartado las entradas y salidas, se hará a continuación, un resumen de todo el sistema para dar una visión general.

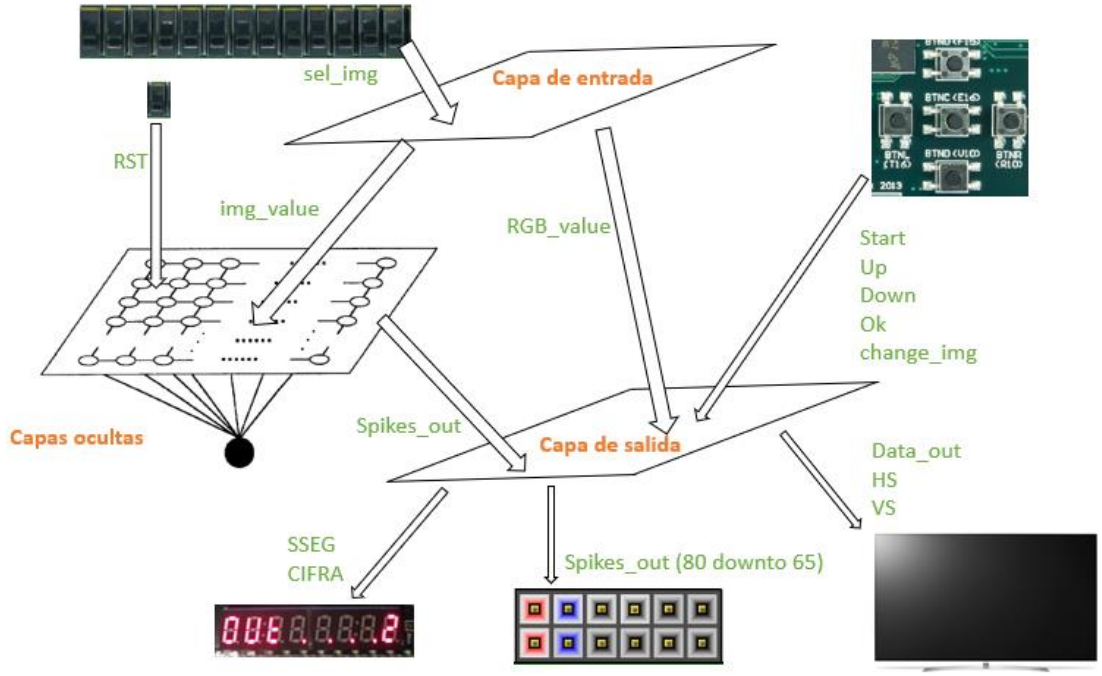


Figura 65 - Sistema electrónico para la realización de redes neuronales LEGION

Tal y como se puede observar en la Figura 65, mediante unos interruptores se puede seleccionar la imagen a analizar. Con esta selección (sel_img), la capa de entrada envía por un lado los valores originales de pixeles (img_value) a la capa oculta; y por el otro lado, los valores de pixeles reducidos (RGB_value) directamente a la capa de salida.

Con los estímulos de entrada (img_value), la LEGION empieza a oscilar acorde a los mecanismos de sincronización y desincronización. Los impulsos producidos de la red (Spikes_out) se envían constantemente a la capa de salida, donde mediante el botón Start, se empieza a buscar los grupos de oscilaciones. Con los botones up y down se selecciona la figura que se desea visualizar, y con ok se visualizará la figura seleccionada en el monitor, conectado mediante un conector VGA. Otras salidas como PMOD, en la cual se podrá visualizar los impulsos de 16 osciladores cuando se conecta con un osciloscopio; y el display de 7 segmentos, dará información del número de figuras encontradas y la figura selecciona actualmente.

La arquitectura en bloques del sistema electrónico diseñado es:

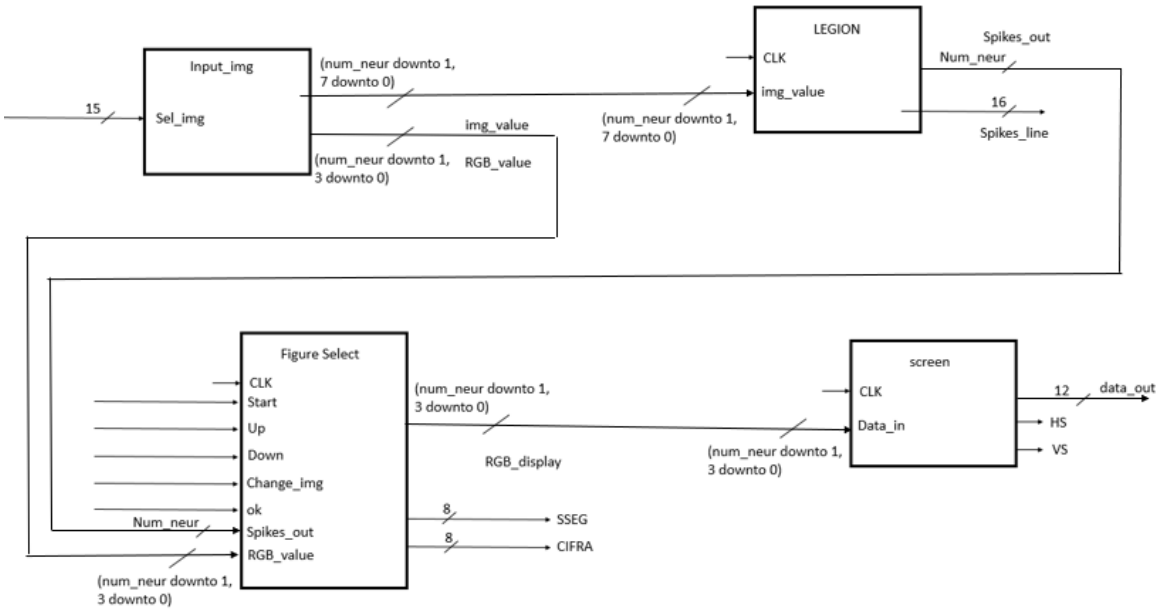


Figura 66 - Esquema de la arquitectura del sistema

7. Reconocimiento de patrones con LEGION

Para comprobar la funcionalidad del sistema diseñado, se le enseña una serie de imágenes, desde las más simples a las con una cierta complejidad. Para visualizar los resultados, se utiliza un monitor BenQ G922GDL para presentar todas las figuras encontradas y un osciloscopio RIGOL DS1102D de 16 entradas digitales para visualizar el estado de los impulsos de los osciladores de la 5ª línea de la imagen.

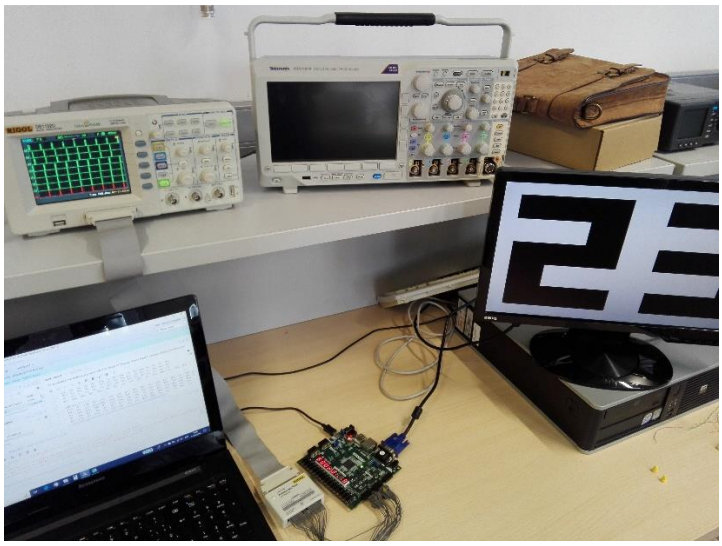


Figura 67 - Toma de resultados experimentales con un monitor y un osciloscopio con entrada digital

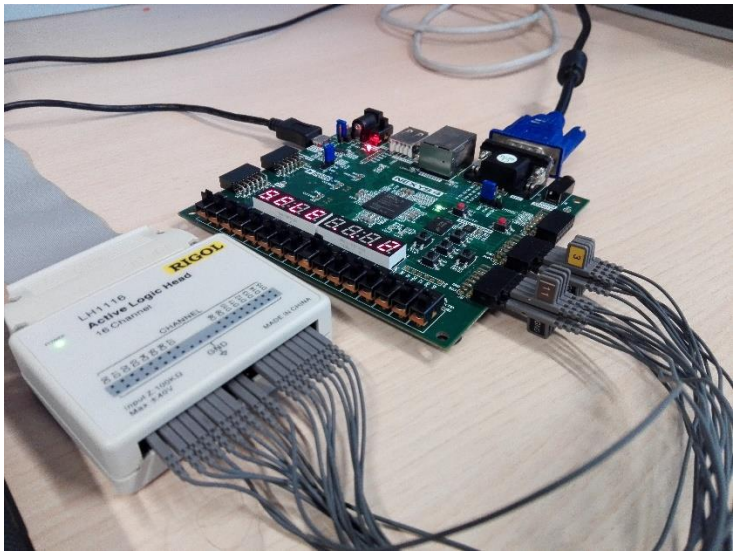


Figura 68 - Conexión del osciloscopio con el puerto PMOD y el monitor con el puerto VGA

7.1. Resultados experimentales en el osciloscopio

Como la red que se implementa para comprobar su funcionalidad es de 192 neuronas, no es posible llevar a la salida todos impulsos de la red ni es posible visualizar todos los impulsos en el osciloscopio al tener una entrada máxima de 16 señales digitales. Por lo que se ha escogida una línea, la 5ª, que está dentro de la imagen para ver su comportamiento a lo largo del tiempo.

Para poder interpretar mejor los resultados obtenidos del osciloscopio, se compara la captura obtenida con la imagen original cuya 5ª línea está marcada por un cuadro verde.

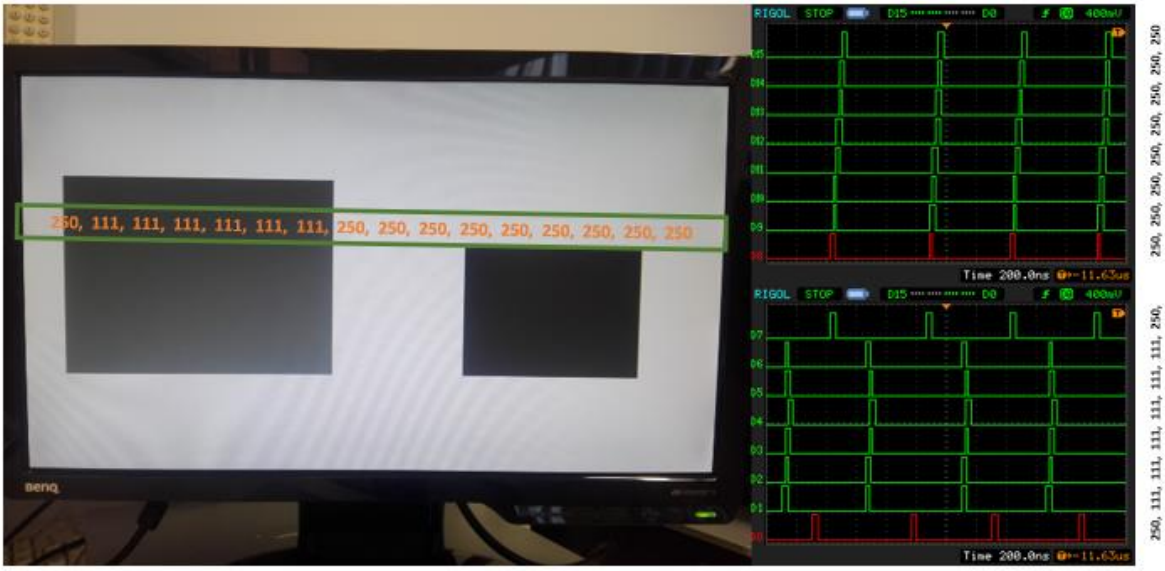


Figura 69 - Impulsos de los osciladores 80 al 65, que corresponden a los valores de pixel de entrada {250, 111, 111, 111, 111, 111, 111, 250, 250, 250, 250, 250, 250, 250, 250, 250}

En la Figura 69 se presentan los impulsos de los osciladores de 65 al 80, cuyos valores de estímulo de entrada (la intensidad de los pixeles) son marcados dentro del cuadro y al lado de la señal correspondiente. Así, la señal D0 (la que está más abajo del todo) representa los impulsos del oscilador 65, el primero de la línea 5, con valor de pixel de entrada 250; la señal D1 representa los impulsos del oscilador 66, el segundo de la línea 5, con valor de pixel de entrada 111; y así sucesivamente. Esta correspondencia se utilizará también para otras imágenes que se presentan a continuación.

Se puede observar que en la 5ª línea de esta figura hay dos figuras, que se refleja también en los impulsos, diferenciando claramente dos grupos de oscilación. Los disparos de D1 a D6 están sincronizados y representan al cuadrado, los disparos de las demás señales están sincronizados también, y representan al fondo.

Aquí, se puede apreciar también el fenómeno de propagación de los impulsos que se observaba en las simulaciones. De un vecino al otro se tarda un ciclo de reloj en inducir el impulso.

Las señales D0 y D7 adelantan un poco a las señales de D8 a D15. Ello no quiere decir que estén saltando como dos grupos de osciladores diferentes, debido a que hay una cantidad muy grande de osciladores que forman la figura del fondo, y los osciladores de D8 al D15 son arrastrados por otros osciladores que están en otras líneas de la figura y algunos de ellos son arrastrados por los osciladores D0 y D7. Al mostrar aquí solo 16 osciladores no se puede tener una visión general de la propagación.

Figura 70 - Impulsos de los osciladores 80 al 65, que corresponden a los valores de pixel de entrada { 250, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 250 }

En la Figura 70 se puede diferenciar claramente dos grupos de osciladores, las señales de D1 a D14 representan al rectángulo y las otras dos señales al fondo.

En la imagen del número 23 (ver Figura 71) se diferencian 3 grupos, dos osciladores D5 y D6 que pertenecen al número 2, dos osciladores D13 y D14 al número 3 y los que quedan, al fondo.

Como se puede observar en los impulsos de cualquier imagen, cuanto mayor es el número de osciladores que forme una figura, más se percibe el efecto de propagación de impulsos.

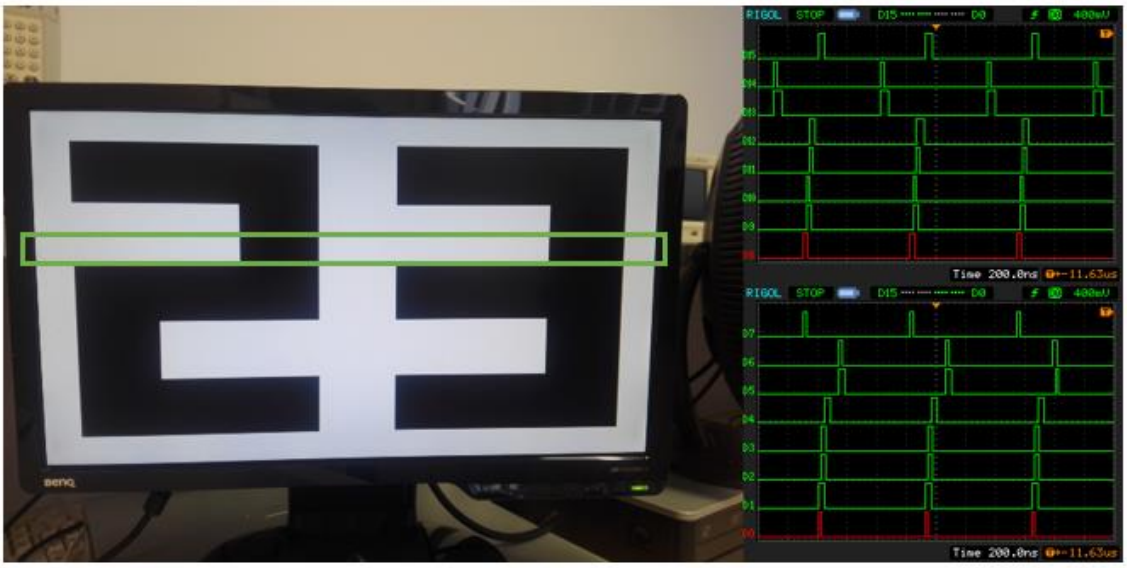


Figura 71 - Impulsos de los osciladores 80 al 65, que corresponden a los valores de pixel de entrada { 250, 250, 250, 250, 250, 10, 10, 250, 250, 250, 250, 250, 250, 10, 10, 250}

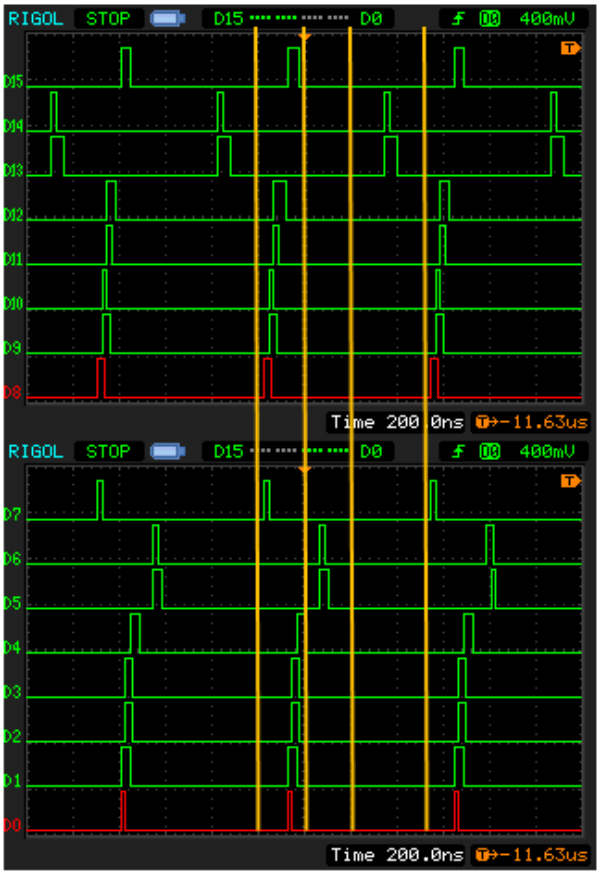


Figura 72 - Impulsos de la Figura 71 aumentada

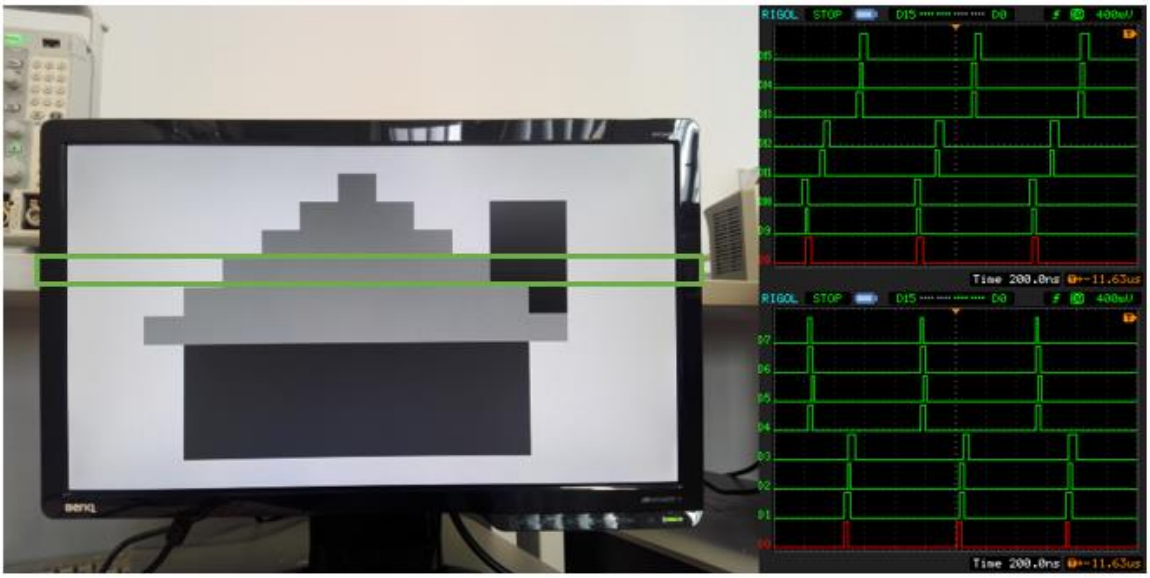


Figura 73 - Impulsos de los osciladores 80 al 65, que corresponden a los valores de pixel de entrada { 250, 250, 250, 250, 150, 150, 150, 150, 150, 150, 150, 10, 10, 250, 250, 250}

En esta imagen de la casa, se puede diferenciar 3 grupos de osciladores sincronizados: un grupo de 7 osciladores (de D4 a D10) que representa al tejado, un grupo de 2 osciladores (D11 y D12) que representa a la chimenea y un último grupo, al fondo.

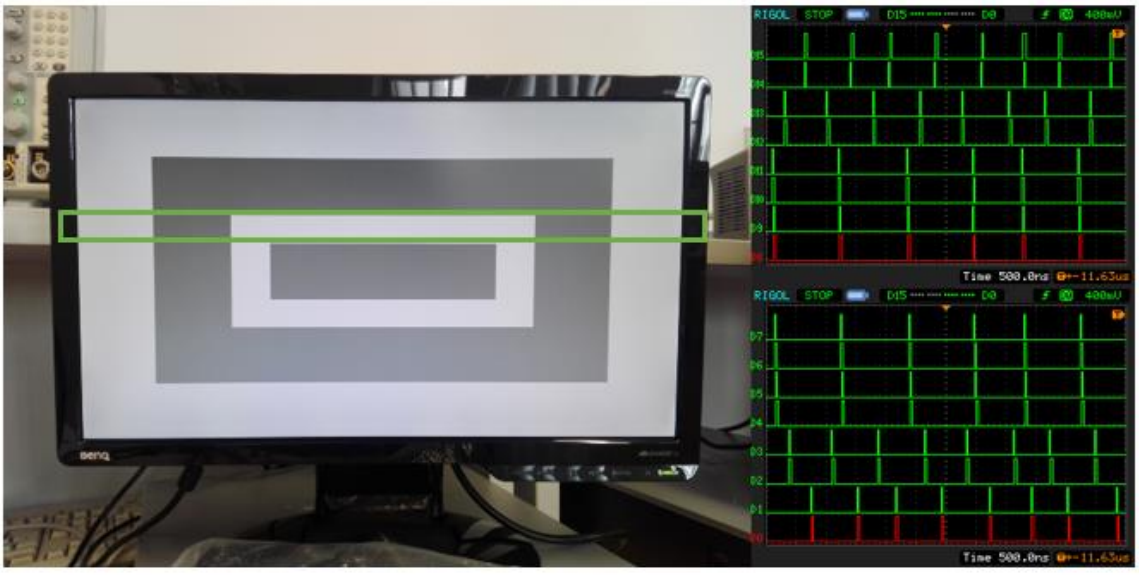


Figura 74 - Impulsos de los osciladores 80 al 65, que corresponden a los valores de pixel de entrada { 250, 250, 150, 150, 250, 250, 250, 250, 250, 250, 250, 250, 150, 150, 250, 250}

Los impulsos de la 5ª línea de esta imagen de rectángulos dentro de rectángulos también indican presencia de 3 grupos sincronizados: el rectángulo blanco (de D4 a D11), el muro negro (D2-D3 y D12-D13) y el fondo (D0-D1 y D14-D15).

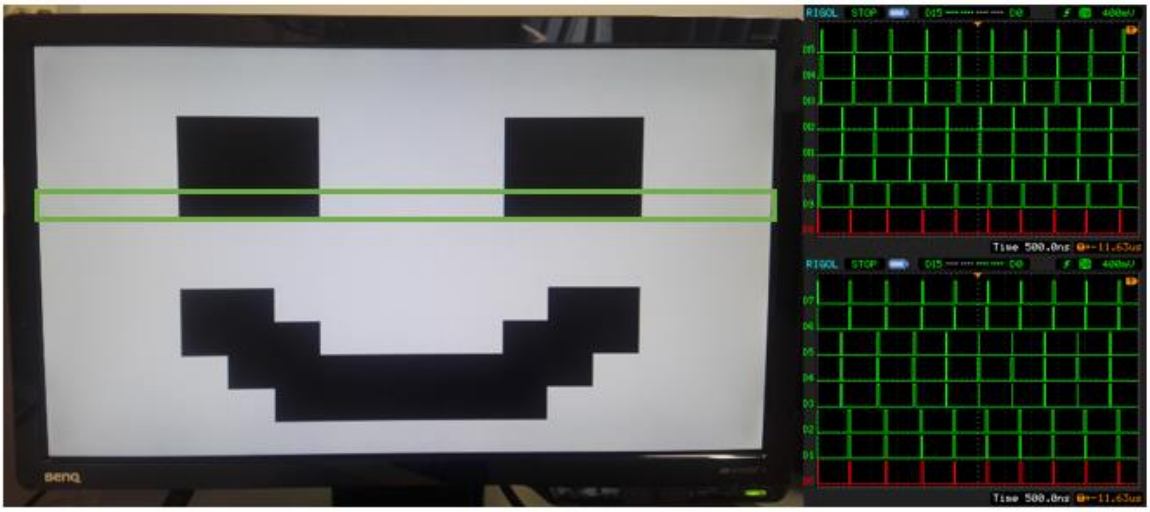


Figura 75 - Impulsos de los osciladores 80 al 65, que corresponden a los valores de pixel de entrada { 250, 250, 250, 10, 10, 10, 250, 250, 250, 250, 10, 10, 10, 250, 250, 250}

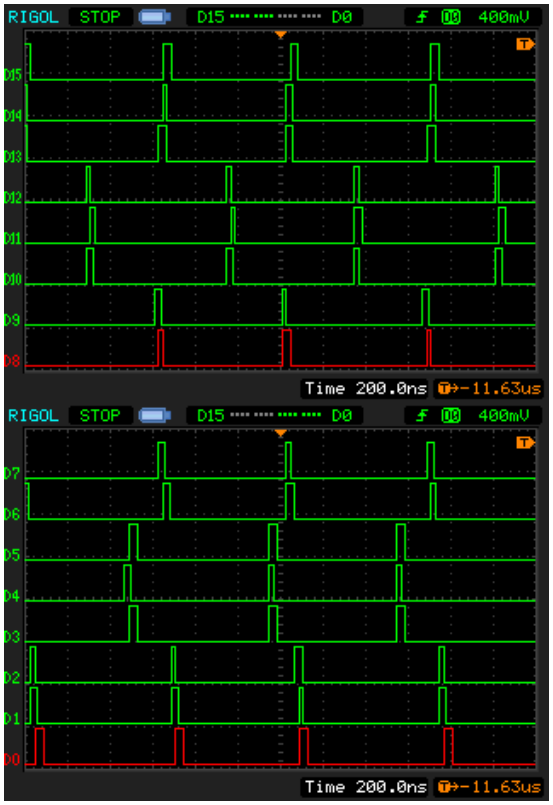


Figura 76 - Impulsos de la Figura 75 aumentada

En la imagen de la sonrisa, se puede diferenciar también 3 grupos de osciladores sincronizados: el ojo izquierdo (de D3 a D5), el ojo derecho (de D10 a D12) y el fondo (de D0 a D2, de D6 a D9 y de D13 a D15).

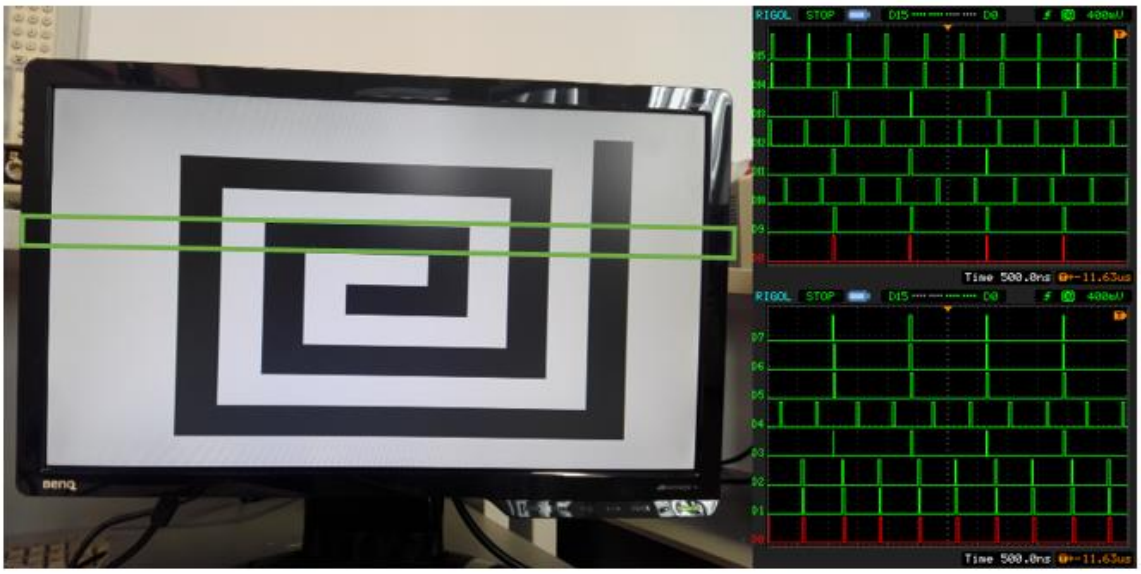


Figura 77 - Impulsos de los osciladores 80 al 65, que corresponden a los valores de pixel de entrada {250, 250, 250, 50, 250, 50, 50, 50, 50, 50, 250, 50, 250, 50, 250, 250}

Figura 78 –Impulsos de la Figura 77 aumentada

Por último, viendo la intensidad de pixel de la 5ª línea, las señales D3, D11 y D13 y las señales de D5 a D9 pertenecen a la espiral y el resto al fondo, tal como se puede comprobar en los impulsos mostrados por el osciloscopio. Además, se puede apreciar que los dos grupos tienen una frecuencia de oscilación diferente.

7.2. Resultados experimentales en el monitor

En todas las imágenes que se presentan a continuación, la primera viñeta es la imagen original (en escala de grises) y las demás son las figuras (en blanco y negro) que el sistema detecta, donde el color negro indica presencia de un impulso y el blanco ausencia de ello.

Empezando por las figuras más simples, cuadrados y rectángulos.

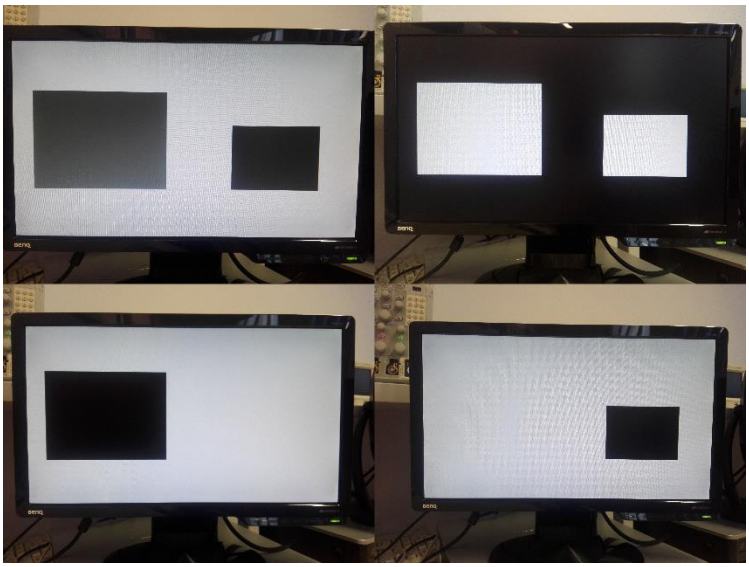


Figura 79 - Reconocimiento de cuadrados en la imagen

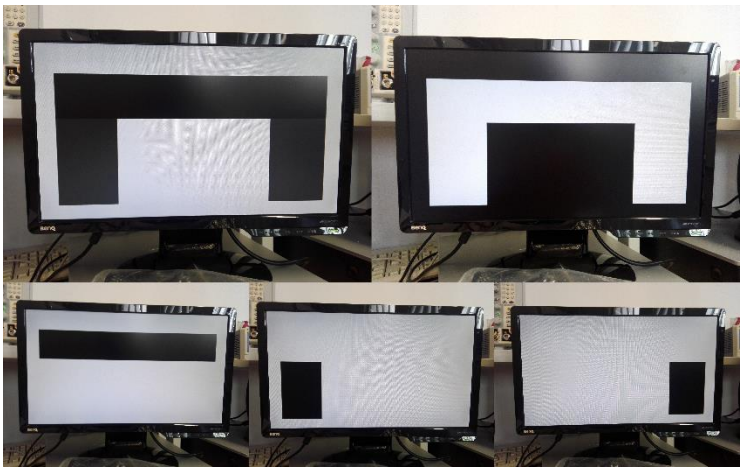


Figura 80 - Reconocimiento de rectángulos en la imagen

Se puede ver en la Figura 79 y en la Figura 80 que la red puede reconocer todas las figuras presentes en la imagen, tanto los dos cuadrados o los tres rectángulos como el fondo.

Si se le enseñan figuras un poco más complicadas, como las resultantes de la combinación de rectángulos y cuadrados (ver Figura 81), o como figuras incrustadas en otras figuras (ver Figura 82), la red reconoce también perfectamente todas las posibles figuras incluyendo el fondo, que equivale a una figura amorfa.

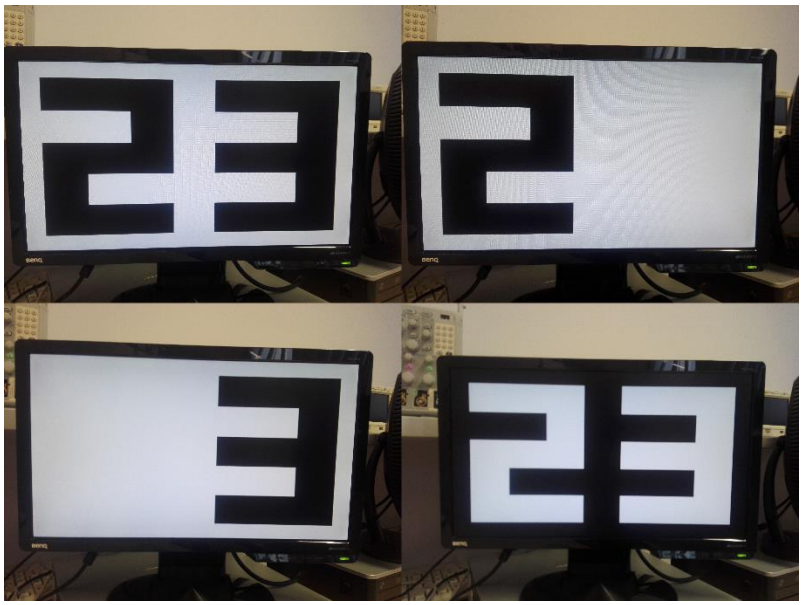


Figura 81 - Reconocimiento de los números 2 y 3 en la imagen

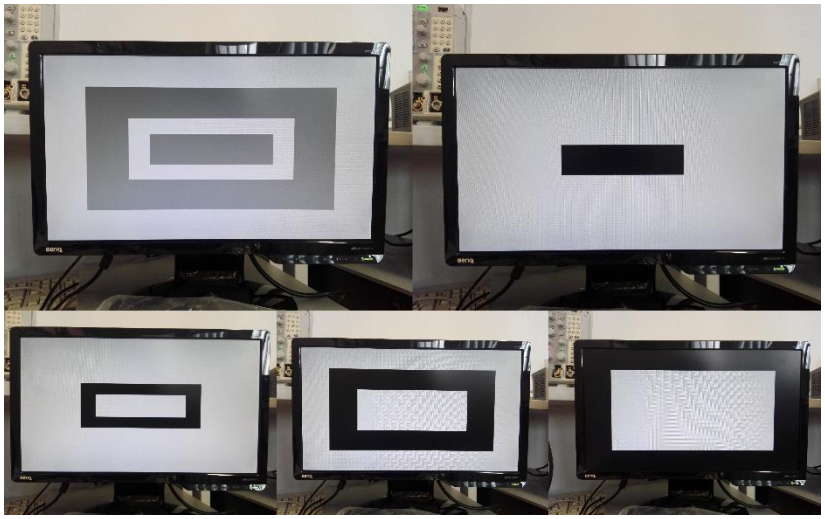


Figura 82 - Reconocimiento de figuras incrustadas en otras figuras

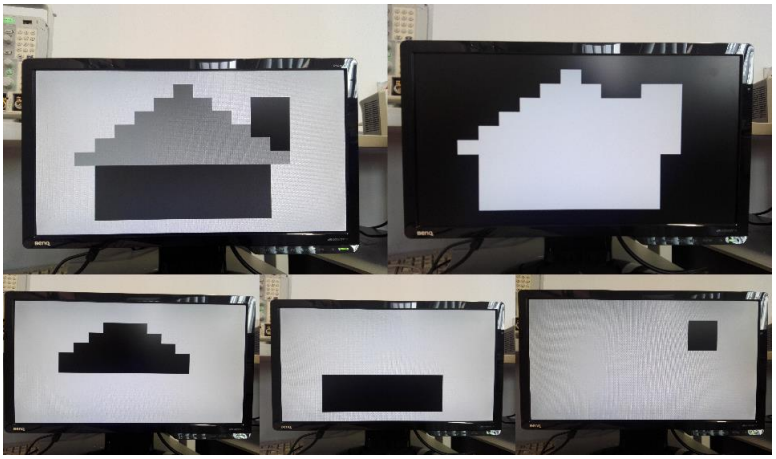


Figura 83 - Reconocimiento de triangulo, trapezoide y rectángulo dentro de una misma imagen

Otras figuras simples, como triángulos o trapezoides, se reconocen con un pequeño defecto. Como se puede observar en la Figura 83, los tres vértices del triángulo no son reconocidos como parte de él. El mismo efecto se observa en el vértice del trapezoide. El fondo y el rectángulo son reconocidos perfectamente. La razón por la que no reconoce estos vértices es que éstos solo tienen un vecino conectado y no reciben fuerza suficiente como para disparar a la misma frecuencia que los demás osciladores del mismo grupo. Por lo que la red los verá como ruido y no formarán parte de ningún grupo oscilador.

Este problema se ve agravado en el monitor porque la resolución de la imagen es de 16x12 pixeles y el defecto es aumentado 40 veces al representarlo en un monitor de 640x480 pixeles. Si la red trabaja con imágenes de mayor resolución, como 640x480, el defecto de que 3 osciladores en un triángulo no salten no se vería en el monitor.

Una vez que se visto que la red puede reconocer figuras de forma simple, se procede a comprobar que pueda reconocer curvas (ver Figura 84) o incluso, espirales (Figura 85).

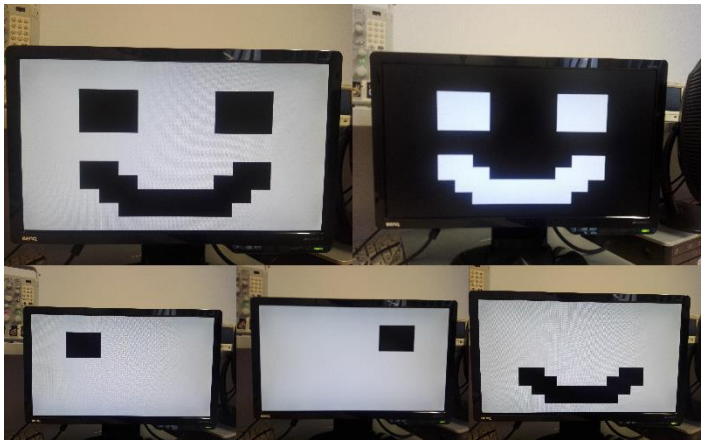


Figura 84 - Reconocimiento de una curva en la imagen

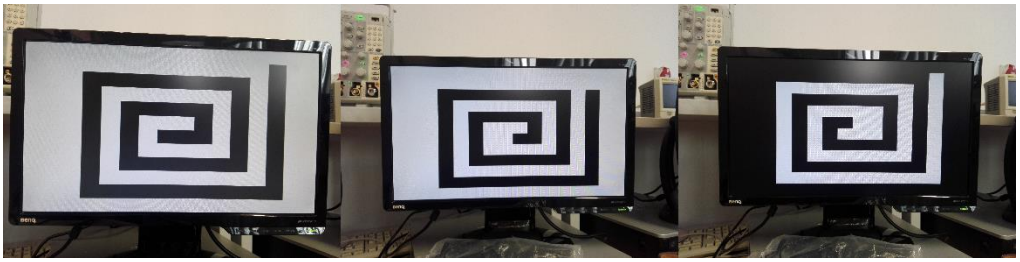


Figura 85 - Reconocimiento de espirales en la imagen

El reconocimiento de la espiral es casi exitoso. Pasa el mismo problema que se mencionaba con los triángulos: los dos extremos de la espiral no son reconocidos porque solo tiene un vecino conectado. En caso de fondo, el pixel que forma extremo de una espiral tampoco es reconocido por la misma razón. Este problema en espirales y curvas no suele ocurrir en práctica, ya que en situaciones normales una espiral o una curva no tienen una anchura de un pixel solo, porque no se verían en la pantalla. Al ser el grosor mayor que un pixel, como en la curva de la Figura 84, los extremos son reconocidos junto con la parte restante.

Conclusiones

El presente trabajo consiste en realizar, en la medida de lo posible, los objetivos establecidos en el inicio del proyecto. Viendo la evolución del proyecto y los problemas que iban surgiendo, se intenta buscar una solución lo más eficiente posible.

Empezando por las teorías de una neurona biológica, se pudo entender el mecanismo de transmisión de información que ésta utiliza. Y se intentó mantener una similitud funcional del sistema nervioso en el sistema electrónico diseñado.

Después de estudiar diferentes posibilidades de modelos de neurona y de redes neuronas artificiales que mejor emulan el comportamiento de la neurona y las redes neuronales biológicas, LEGION ha destacado para una realización flexible de redes neuronales que emulan el córtex visual. Su interconexión reducida y no necesidad de un mecanismo de aprendizaje reducen el uso de recursos haciéndolo más apto para una implementación a muy gran escala (VLS), muy interesante si se quiere implementar una red de tamaño elevado y, por lo tanto, poder resolver mayor número de tareas de mayor complejidad (por ejemplo, analizar 3 imágenes a la vez).

La dificultad en implementar LEGION reside en que el modelo original del oscilador es muy complejo, por lo que se procedió a su simplificación, basándose en el modelo Integrate-and-Fire modificado por B. Girau y C. Torres-Huitzil. Aunque la red LEGION modificada combina características de ambas redes LEGION y SNN, ofreciendo un mayor realismo que LEGION original y menor complejidad que SNN, las posibilidades de su aplicación se reducen al campo de visión artificial, concretamente reconocimiento de figuras en la imagen en este caso.

Y viendo los resultados obtenidos, se hizo otras adaptaciones de la ecuación y se ajustaron los parámetros de peso y de inhibición, cruciales para el correcto funcionamiento de la red. Además, se han solucionado los problemas de timing, para poder implementar con éxito el diseño en el FPGA.

Para poder visualizar los resultados del reconocimiento, se diseñaron unos circuitos a la salida de la red que adaptan los impulsos de salida en señales de información RGB para un monitor. Y se conectan impulsos de 16 osciladores de la red con un osciloscopio con entradas digitales mediante el puerto PMOD para visualizar el estado y la frecuencia de disparo de estos impulsos. Además, se ofrece una interface que interactúe con el usuario: posibilidad de seleccionar imágenes a analizar y la figura a visualizar.

Y para comprobar la funcionalidad del sistema electrónico diseñado, se le enseña una serie de imágenes prediseñadas. Tanto en las simulaciones como en los resultados experimentales se pudo observar que los diferentes grupos de osciladores representan a figuras presentes en la imagen,

comprobando la teoría de sincronismo de LEGION. El reconocimiento de diferentes figuras resulta satisfactorio, desde figuras simples como cuadrados y rectángulos, a combinación de figuras simples e incluso es capaz de reconocer con éxito curvas y espirales. El único defecto es que no puede reconocer los vértices de un triángulo, puntos agujados de una figura, ni los dos extremos de una curva cuando su grosor es de un pixel solo. Este defecto no es relevante en práctica, ya que 3 pixeles en un triángulo es casi imperceptible y no es habitual curvas de un pixel de grosor.

Otro punto destacado de la red diseñada es que ésta es escalable y flexible, puede analizar imágenes de diferente resolución cambiando los parámetros genéricos {im_length, im_width, num_neur}, siempre que el número total de pixeles sea menor que 550, si se utiliza un FPGA de familia Artix7 con modelo XC7A100T-CSG324. Y es posible conectar con la red entradas (otras imágenes) o dispositivos de salida diferentes de los propuestos, cambiando solamente la capa de entrada o de salida dejando la red intacta gracias al diseño modular.

La red diseñada cumple, además, con los alcances definidos desde el principio: tiene un tamaño mayor de 100 neuronas, tiene un interfaz que interactué con el usuario ofreciendo control para visualizar los resultados, tiene tolerancia a interferencias (la unidad LFSR actúa como una fuente constante de ruido) y a fallos.

Para que cualquier individuo pueda utilizar este sistema diseñado, se provee un manual de usuario en el anexo donde se explica, entre otras cosas, las variables más importantes para modificar el sistema diseñado y completar tareas de reconocimiento de figuras en imágenes de diferente resolución que la presentada como ejemplo en este trabajo.

Finalmente, este trabajo sirve de base para trabajos futuros. Una posibilidad sería utilizar la salida de una cámara como la entrada de imagen, mediante el puerto VGA o USB. Otra posibilidad es en unir varios dispositivos FPGA para poder trabajar con imágenes de mayor tamaño o con varias imágenes a la vez. Una tercera posibilidad es en construir una LEGION de mayor dimensión, incorporando otra dimensión perpendicular al plano de la imagen, de esta forma se podrá reconocer patrones en objetos en el espacio.

Presupuesto

A continuación, se hace una estimación del coste para poder llevar a cabo todo el trabajo, teniendo en cuenta el coste de los equipos utilizados, la licencia del software utilizado y las horas invertidas en el desarrollo del trabajo.

En primer lugar, los equipos utilizados en este trabajo son: un portátil personal para diseñar, sintetizar y simular todo el diseño, escribir la presente memoria y dibujar los esquemas y los diagramas de bloques; una placa FPGA Nexys4 DDR Artix-7 donde el diseño es implementado para demostrar la funcionalidad del sistema electrónico; un monitor que permita visualizar los resultados del reconocimiento de figuras; un osciloscopio analógico que permite capturar la forma del impulso de un oscilador y un osciloscopio digital que permite capturar el estado del impulso (on/off) de una línea de osciladores en la red.

Tabla 2 - Estimación del coste de los equipos utilizados

	Precio original (€)	*Vida útil (año)*	*Tiempo utilizado (meses)*	*Precio equivalente (€)*
Portátil Lenovo	650,00	5	5	54,17
Nexys4 DDR Artix-7	262,85	5	5	21,90
Tektronix MDO3034	9970,40	10	3	249,26
BenQ G922HDL	48,50	5	3	2,43
Rigol DS1102D	679,07	10	1	5,66
Total	**11610,82**	**--**	**--**	**333,42**

En segundo lugar, se ha utilizado una serie de herramientas de software. Xilinx Vivado fue la principal herramienta utilizada en el diseño y simulación del sistema, cuya licencia de estudiantes, WebPack License, es gratuita. Para la elaboración de la presente memoria se utilizó el software Word y PowerPoint de Microsoft Office. Y el sistema operativo del ordenador portátil utilizado en este trabajo es gratuito también, viniendo de la fábrica.

Tabla 3 - Estimación de los costes de software utilizado

	Precio por mes (€)	*Tiempo utilizado (mes)*	*Precio final*
WebPack License	0	5	0
Microsoft Office	7,00	5	35,00
Windows 10	0	5	0
Total	**7,00**	**5**	**35,00**

El último tipo de coste para tener en cuenta son las horas trabajadas para realizar este proyecto. Según el convenio entre la universidad UPC y las empresas en cooperación para ofrecer prácticas a estudiantes, el salario orientativo recomendado por la universidad es de 8 €/hora.

Tabla 4 - Estimación del salario

	Horas trabajadas	*Precio por hora*	*Total*
Salario del estudiante	600	8	4800,00

Sumando todos los costes calculados anteriormente, el proyecto tendría un coste de desarrollo de 5168,42 euros.

Tabla 5 - Coste total para desarrollar el proyecto

Tipos de coste	*Precio (€)*
Hardware y equipos de medida	333,42
Software	35,00
Salario	4800,00
Total	**5168,42**

Bibliografía

[1] E. Solomon, L. Berg, and D. Martin, *Biología*, Novena Edi. México: Cengage Learning Editores, S.A. de C.V., 2013.

[2] A. Rodríguez, "Neurona," *Biopsicosalud*, 2017. [Online]. Available: http://www.biopsicosalud.com.ve/2017/05/neurona.html. [Accessed: 24-May-2018].

[3] "1.2.-Fisiologia del musculo cardiaco | Volviendo a lo básico," *Fundación para la Formación e Investigacón Sanitarias de la Región de Murcia*. [Online]. Available: http://www.ffis.es/volviendoalobasico/12fisiologia_del_musculo_cardiaco.html. [Accessed: 24-May-2018].

[4] D. J. Matich, "Redes Neuronales: Conceptos Básicos y Aplicaciones.," Universidad Tecnológica Nacional, 2001.

[5] Kiyoshi Kawaguchi, "The McCulloch-Pitts Model of Neuron," *The University of Texas at El Paso (UTEP)*, 2000. [Online]. Available: http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node12.html. [Accessed: 24-Mar-2018].

[6] D. Heeger, "Integrate and Fire Model of Spike Generation," *J. Neurosci.*, pp. 1–4, 1997.

[7] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, vol. 1. 2002.

[8] a L. Hodgkin and a F. Huxley, "A quantitative description of membrane current and its applicaiton to conduction and excitation in nerve," *J Physiol*, vol. 117, no. 4. pp. 500–544, 1952.

[9] M. Nelson and J. Rinzel, "The Hodgkin-Huxley Model," *Genesis*, vol. 125, no. 20, pp. 29–50, 1990.

[10] R. Hecht-nielsen, "Perceptrons," *Univ. Calif. San Diego Inst. Neural Comput.*, no. 0403, pp. 1–59, 2004.

[11] R. Q. Juan and a Chacón, "Redes neuronales artificiales para el procesamiento de imágenes , una revisión de la última década," *Riee&C*, vol. 9, no. 1, pp. 7–16, 2011.

[12] F. Ponulak and A. Kasiński, "Introduction to spiking neural networks : Information processing , learning and applications," *Acta Neurobiol. Exp. (Wars).*, vol. 71, pp. 409–433, 2011.

[13] E.-G. M. Mallorquí, "DIGITAL SYSTEM FOR SPIKING NEURAL NETWORK," Universitat Politècnica de Catalunya, 2017.

[14] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in …," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.

[15] S. Haykin, *Neural Networks and Learning Machines*, Third Edit. New Jersey: Pearson Education, Inc., 2008.

[16] S. I. Gallant, “Perceptron-Based Learning Algorithms,” *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 179–191, 1990.

[17] H. J. David Velasquez, “Acotación del error de modelos de redes neuronales aplicados al pronóstico de series de tiempo.,” *UIS Ing.*, vol. 10, no. 1, pp. 63–69, 2011.

[18] D. Wang and D. Terman, “Locally Excitory Globally Inhibitory Oscillator Networks,” *IEEE Trans. Neural Networks*, vol. 6, no. 1, pp. 283–286, 1995.

[19] D. L. Wang, “Emergent Synchrony in Locally Coupled Neural Oscillators,” *IEEE Trans. Neural Networks*, vol. 6, no. 4, pp. 941–948, 1995.

[20] B. Girau and C. Torres-Huitzil, “FPGA implementation of an integrate-and-fire legion model for image segmentation,” *Eur. Symp. Artif. Neural*, no. April, pp. 26–28, 2006.

[21] R. Guerra, “Escala de grises y tramas | Xilografía Concón,” *Xilografía Concón*. [Online]. Available: https://xilografiaconcon.wordpress.com/2014/06/06/escala-de-grises-y-tramas/. [Accessed: 20-May-2018].

[22] N. W. Instruments, “LFSR Reference -- M-Sequence, Linear Feedback Shift Register, Feedback Taps for Maximal Length Sequences,” 2010. [Online]. Available: http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm. [Accessed: 22-May-2018].

[23] A. Morro, V. Canals, A. Oliver, M. L. Alomar, and J. L. Rossello, “Ultra-fast data-mining hardware architecture based on stochastic computing,” *PLoS One*, vol. 10, no. 5, p. e0124176, May 2015.

[24] “AR# 1072: Fast outputs versus fast slew rate outputs in Xilinx devices,” *Forum Xilinx*. [Online]. Available: https://www.xilinx.com/support/answers/1072.html. [Accessed: 01-Jun-2018].

[25] Tektronix, “MDO4000 Series Mixed Domain Oscilloscopes.” [Online]. Available: https://www.tek.com/oscilloscope/mdo3000-mixed-domain-oscilloscope. [Accessed: 02-Jun-2018].

[26] Digilent, “Nexys 4™ FPGA Board Reference Manual.” pp. 1–29, 2016.

Anexos

A1. Manual de usuario

1. Introducción

Este manual de usuario tiene objetivo de ser utilizado por cualquier individuo interesado en desarrollar unas redes neuronales artificiales que sean capaces de reconocer figuras o patrones dentro de una imagen utilizando el sistema electrónico presentado en este trabajo. Para ello, es imprescindible disponer del paquete de software Xilinx Vivado, con un mínimo requisito de licencia (WebPack License), al ser el diseño integradamente en VHDL.

En Vivado, se utilizan 3 tipos diferentes de archivos:

- Design sources: archivos de diseño escritos en VHDL. Son los que definen la estructura y la funcionalidad del sistema
- Simulation sources: archivos de simulación escritos en VHDL. Su función es la de simular el comportamiento del sistema electrónico descrito utilizando Run Simulation del programa.
- XDC Constraints: archivos donde se definen la ubicación de los puertos de entrada y salida en la placa FPGA, la tensión de salida de los puertos, el retraso de las señales de entrada y salida respecto del reloj, etc.

A continuación, se explicará cómo utilizar los archivos diseñados para poder ser utilizado por el tercero.

2. Design sources

Unas instrucciones generales sobre el uso de diferentes archivos de diseño son proporcionadas en esta sección. El sistema diseñado es jerárquico, donde la entidad del nivel más alto (TOP.vhd) engloba a los 4 bloques que forman el sistema: Neur_Net.vhd, Figure_Select.vhd, display_screen.vhd y input_img.vhd. Éstos, a su vez, son formados por unidades de menor nivel.

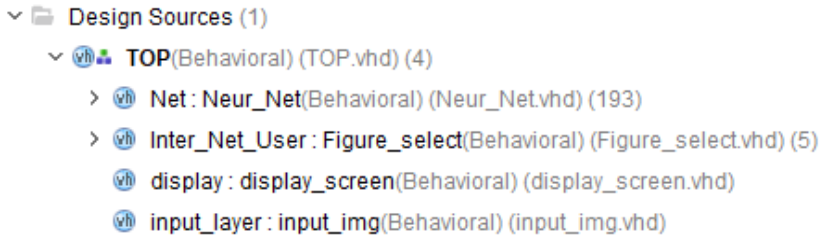


Figura 86 - Organización de los niveles del diseño

2.1. Entidad TOP

En el archivo TOP.vhd se definen todas las interconexiones de los bloques que forman el sistema y las conexiones de entrada y salida con el sistema, además de los parámetros que se utilizan para controlar el tamaño de la red y para definir ciertas características que influyen en el procesamiento de las imágenes.

Dichos parámetros se encuentran dentro de la definición de la entidad TOP, en Generics. A continuación, se hace una descripción sobre cada uno de ellos:

Tabla 6 - Generics de la entidad TOP

Generic	*Descripción*	*Valor por defecto*
Pixel_size	Numero de bits para definir la intensidad del pixel de una imagen en escala de grises	8
Threshold	Diferencia en intensidad de pixel entre dos pixeles vecinos para ser considerados de formar la misma figura, para su cálculo hay que tener en cuenta pixel_size	16
Num_neur	Número total de pixeles que forman la imagen	192
Im_length	Número de pixeles en una fila de la imagen	16
Im_width	Número de pixeles en una columna de la imagen	12

De esta forma, si se quiere trabajar con una imagen de 32x26, ha de cambiar im_length por 32, im_width por 26 y num_neur por 32x26=832. Y si se quiere trabajar con intensidades de pixel definidas con 15 bits y considerar un criterio de 5% de diferencia, pixel_size ha de establecer en 15 y threshold en 5% · (2^15) = 1638,4 ≈ 1638.

El sistema electrónico diseñado es tan simple de manejar que no necesita cambiar ningún parámetro más de la entidad o de los bloques que la forman, excepto la imagen de entrada.

2.2. Entidad input_img

Si se quiere trabajar con otras imágenes distintas de las presentes en este archivo, se ha de definir la nueva imagen modificando el array de vectores de cualquier imagen ya presente. Un ejemplo de este array es el presentado en el apartado 6.1.1. Ten en cuenta que los valores de deben estar dentro del rango definido por pixel_size.

3. Simulation sources

Tres archivos de simulación son proporcionados como plantillas por defecto: uno que simula el comportamiento de todo el sistema electrónico, otro que simula solamente el comportamiento de la red neuronal, y el último, que simula la interfaz de interacción con el usuario.

Los parámetros generic son los mismo explicados para la entidad TOP en design sources.

4. XDC constraints

Las restricciones o constraints provenidos en este trabajo son específicos para un FPGA de familia Artix7, en concreto, Nexys4 DDR Trainer Board, o modelo XC7A100T-CSG324. Si se implementa el sistema electrónico en cualquier otro dispositivo, consulte al manual de usuario de su fabricante.

En este archivo, constraints_TOP.xdc, se definen la localización de todas las entradas y salidas en la placa FPGA. Se definen, además, retraso de éstas respecto a la señal de reloj, para cumplir con los tiempos de set-up y hold-off.

A2. Código en VHDL de la LEGION

1. Design sources

1.1.TOP.vhd

```
----------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 09.05.2018 12:04:45
-- Design Name:
-- Module Name: TOP - Behavioral
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.bus_multiplexer_pkg.all;

entity TOP is
    Generic (  threshold: integer:= 16;
               pixel_size: integer := 8;
               num_neur: integer := 192;
               im_length: integer := 16;
               im_width: integer := 12);

    Port (  CLK: in STD_LOGIC;
            start : in STD_LOGIC;
            up : in STD_LOGIC;
            down : in STD_LOGIC;
            ok : in STD_LOGIC;
            change_img: in STD_LOGIC;
            RST: in STD_LOGIC;
            sel_img: in std_logic_vector(14 downto 0);
            spikes_line: out STD_LOGIC_VECTOR(15 downto 0);
            DATA_OUT : out STD_LOGIC_VECTOR (11 downto 0);
            HS : out STD_LOGIC;
            VS : out STD_LOGIC;
            SSEG : out STD_LOGIC_VECTOR (7 downto 0);
            CIFRA : out STD_LOGIC_VECTOR (7 downto 0));
end TOP;

architecture Behavioral of TOP is

component input_img is
    Generic(    num_neur: natural;
                pixel_size: natural);
    Port ( sel_img : in STD_LOGIC_VECTOR (14 downto 0);
           img_value : out bus_array(num_neur downto 1, pixel_size-1
downto 0);
           RGB_value: out bus_array(num_neur downto 1, 3 downto 0));
end component;

component Neur_Net is
    Generic (  threshold: integer;
```

```
                num_neur: integer;
                pixel_size: integer;
                im_length: integer;
                im_width: integer);
    Port (  CLK: in STD_LOGIC;
            RST: in STD_LOGIC;
            img_value : in bus_array(num_neur downto 1, pixel_size-1
downto 0);
            spikes_line: out STD_LOGIC_VECTOR(15 downto 0);
            Spikes_out : out STD_LOGIC_VECTOR (num_neur downto 1));

end component;

component Figure_select is
    Generic( num_neur: integer;
             pixel_size: integer);
    Port ( spikes_out : in STD_LOGIC_VECTOR (num_neur downto 1);
           RGB_value: in bus_array(num_neur downto 1, 3 downto 0);
           CLK:  in STD_LOGIC;
           start : in STD_LOGIC;
           up : in STD_LOGIC;
           down : in STD_LOGIC;
           ok : in STD_LOGIC;
           change_img: in STD_LOGIC;
           RGB_display: out bus_array(num_neur downto 1, 3 downto 0);
           SSEG : out STD_LOGIC_VECTOR (7 downto 0);
           CIFRA : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component display_screen is
    Generic(    im_length: integer;
                im_width: integer;
                num_neur : integer);
    Port ( CLK : in STD_LOGIC;
           DATA_IN : in bus_array(num_neur downto 1, 3 downto 0);
           DATA_OUT : out STD_LOGIC_VECTOR (11 downto 0);
           HS : out STD_LOGIC;
           VS : out STD_LOGIC);
end component;

signal spikes: STD_LOGIC_VECTOR (num_neur downto 1) := (others => '0');
signal im_aux: bus_array(num_neur downto 1, pixel_size-1 downto 0);
signal RGB_aux, RGB_out:bus_array(num_neur downto 1, 3 downto 0);
signal spikes_line_aux: STD_LOGIC_VECTOR(15 downto 0);
signal RST_aux: std_logic;

begin

Net:Neur_Net
generic map(    pixel_size => pixel_size,
                num_neur => num_neur,
                threshold => threshold,
                im_length => im_length,
                im_width => im_width)
port map(       CLK => CLK,
                RST => RST_aux,
                img_value => im_aux,
                spikes_line => spikes_line_aux,
```

```
                spikes_out => spikes);

Inter_Net_User: Figure_select
generic map(    num_neur => num_neur,
                pixel_size => pixel_size)
port map(       CLK => CLK,
                start => start,
                up => up,
                down => down,
                ok => ok,
                change_img => change_img,
                spikes_out => spikes,
                RGB_value => RGB_aux,
                RGB_display => RGB_out,
                SSEG => SSEG,
                CIFRA => CIFRA);

display: display_screen
generic map(    im_length => im_length,
                im_width => im_width,
                num_neur => num_neur)
port map(       CLK => CLK,
                HS => HS,
                VS => VS,
                DATA_IN => RGB_out,
                DATA_OUT => DATA_OUT);

input_layer: input_img
generic map(    num_neur => num_neur,
                pixel_size => pixel_size)
port map(       sel_img => sel_img,
                img_value => im_aux,
                RGB_value => RGB_aux);

process(CLK)
begin
    if rising_edge(CLK) then
        spikes_line <= spikes_line_aux;
        RST_aux <= RST;
    end if;
end process;
end Behavioral;
```

1.2.Neur_Net.vhd

```
----------------------------------------------------------------------------
---------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 17.04.2018 12:48:13
-- Module Name: Neur_Net - Behavioral
----------------------------------------------------------------------------
---------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.bus_multiplexer_pkg.all;

entity Neur_Net is
    Generic (  threshold: integer:= 16;
               pixel_size: integer := 8;
              num_neur: integer := 192;
              im_length: integer := 16;
              im_width: integer := 12);
    Port ( CLK: in STD_LOGIC;
           RST: in STD_LOGIC;
           img_value : in bus_array(num_neur downto 1, pixel_size-1
downto 0);
           spikes_line: out STD_LOGIC_VECTOR(15 downto 0);
           Spikes_out : out STD_LOGIC_VECTOR (num_neur downto 1));
end Neur_Net;

architecture Behavioral of Neur_Net is

component Neur_Unit is
    generic ( pixel_size: integer;
            Zi : integer;
            current_neur: integer;
            num_neur: integer;
            threshold: integer;
            has_up, has_down, has_left, has_right: boolean);
    Port ( CLK : in STD_LOGIC;
           P1,P2,P3,P4: in STD_LOGIC;
           pixel_up : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           pixel_down : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           pixel_left : in STD_LOGIC_VECTOR (pixel_size -1  downto 0);
           pixel_right : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           pixel_i : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           G : in STD_LOGIC;
           RST: in STD_LOGIC;
           spike_out: out STD_LOGIC);
end component;

component inh is
    Generic( num_neur: integer);
    Port ( Spikes_in : in STD_LOGIC_VECTOR (num_neur downto 1);
           G : out STD_LOGIC);
end component;

-- read input image signals--
```

```
type im is array (num_neur downto 1) of std_logic_vector(pixel_size -1
downto 0);
signal im_value: im := (others => (others => '0'));

---- internal signals
signal spikes: std_logic_vector(num_neur downto 1):=(others=>'0');
signal G : STD_LOGIC;

begin

-- convertion of the bidimensional array input into an unidimensional
array --
process(CLK)
begin
    if rising_edge(CLK) then
        for j in 1 to num_neur loop
            for k in 0 to pixel_size-1 loop
                im_value(j)(k) <= img_value(j,k);
            end loop;
        end loop;
    end if;
end process;


-- connection between the neurons
Neuron: for i in 1 to num_neur generate

    upper_edge: if ((i < im_length) and i/=1) generate
        upper_neur: Neur_Unit
            generic map(pixel_size => pixel_size,
                        Zi => 3,
                        current_neur => i,
                        num_neur => num_neur,
                        threshold => threshold,
                        has_up => false,
                        has_down => true,
                        has_left => true,
                        has_right => true)
            port map(  CLK => CLK,
                       P1 => '0',
                       P2 => spikes(i + im_length),
                       P3 => spikes(i-1),
                       P4 => spikes(i+1),
                       pixel_up => (others => '0'),
                       pixel_down => im_value(i + im_length),
                       pixel_left => im_value(i-1),
                       pixel_right => im_value(i+1),
                       pixel_i => im_value(i),
                       G => G,
                       RST => RST,
                       spike_out => spikes(i));
    end generate upper_edge;

    down_edge: if ((i > (im_width-1)*im_length +1) and (i /= num_neur))
generate
        down_neur: Neur_Unit
            generic map(pixel_size => pixel_size,
                        Zi => 3,
```

```
                        current_neur => i,
                        num_neur => num_neur,
                        threshold => threshold,
                        has_up => true,
                        has_down => false,
                        has_left => true,
                        has_right => true)
            port map(   CLK => CLK,
                        P1 => spikes(i - im_length),
                        P2 => '0',
                        P3 => spikes(i-1),
                        P4 => spikes(i+1),
                        pixel_up => im_value(i - im_length),
                        pixel_down => (others => '0'),
                        pixel_left => im_value(i-1),
                        pixel_right => im_value(i+1),
                        pixel_i => im_value(i),
                        G => G,
                        RST => RST,
                        spike_out => spikes(i));
    end generate down_edge;

    left_edge: if (((i mod im_length) = 1) and (i /= 1) and (i /=
(num_neur - im_length +1))) generate
        left_neur: Neur_Unit
            generic map(pixel_size => pixel_size,
                        Zi => 3,
                        current_neur => i,
                        num_neur => num_neur,
                        threshold => threshold,
                        has_up => true,
                        has_down => true,
                        has_left => false,
                        has_right => true)
            port map(   CLK => CLK,
                        P1 => spikes(i - im_length),
                        P2 => spikes(i + im_length),
                        P3 => '0',
                        P4 => spikes(i+1),
                        pixel_up => im_value(i - im_length),
                        pixel_down => im_value(i + im_length),
                        pixel_left => (others => '0'),
                        pixel_right => im_value(i+1),
                        pixel_i => im_value(i),
                        G => G,
                        RST => RST,
                        spike_out => spikes(i));
    end generate left_edge;

    right_edge: if  ((i mod im_length = 0) and (i/=im_length) and
(i/=num_neur)) generate
        right_neur: Neur_Unit
            generic map(pixel_size => pixel_size,
                        Zi => 3,
                        current_neur => i,
                        num_neur => num_neur,
                        threshold => threshold,
                        has_up => true,
```

```
                    has_down => true,
                    has_left => true,
                    has_right => false)
        port map(   CLK => CLK,
                    P1 => spikes(i - im_length),
                    P2 => spikes(i + im_length),
                    P3 => spikes(i-1),
                    P4 => '0',
                    pixel_up => im_value(i - im_length),
                    pixel_down => im_value(i + im_length),
                    pixel_left => im_value(i-1),
                    pixel_right => (others => '0'),
                    pixel_i => im_value(i),
                    G => G,
                    RST => RST,
                    spike_out => spikes(i));
end generate right_edge;

left_upper_corner: if (i = 1) generate
    corner1: Neur_Unit
        generic map(pixel_size => pixel_size,
                    Zi => 2,
                    current_neur => i,
                    num_neur => num_neur,
                    threshold => threshold,
                    has_up => false,
                    has_down => true,
                    has_left => false,
                    has_right => true)
        port map(   CLK => CLK,
                    P1 => '0',
                    P2 => spikes(i + im_length),
                    P3 => '0',
                    P4 => spikes(i+1),
                    pixel_up => (others => '0'),
                    pixel_down => im_value(i + im_length),
                    pixel_left => (others => '0'),
                    pixel_right => im_value(i+1),
                    pixel_i => im_value(i),
                    G => G,
                    RST => RST,
                    spike_out => spikes(i));
end generate left_upper_corner;

right_upper_corner: if (i = im_length) generate
    corner2: Neur_Unit
        generic map(pixel_size => pixel_size,
                    Zi => 2,
                    current_neur => i,
                    num_neur => num_neur,
                    threshold => threshold,
                    has_up => false,
                    has_down => true,
                    has_left => true,
                    has_right => false)
        port map(   CLK => CLK,
                    P1 => '0',
                    P2 => spikes(i + im_length),
```

```
                    P3 => spikes(i-1),
                    P4 => '0',
                    pixel_up => (others => '0'),
                    pixel_down => im_value(i + im_length),
                    pixel_left => im_value(i-1),
                    pixel_right => (others => '0'),
                    pixel_i => im_value(i),
                    G => G,
                    RST => RST,
                    spike_out => spikes(i));
end generate right_upper_corner;

left_down_corner: if (i = (num_neur - im_length +1)) generate
    corner3: Neur_Unit
        generic map(pixel_size => pixel_size,
                    Zi => 2,
                    current_neur => i,
                    num_neur => num_neur,
                    threshold => threshold,
                    has_up => true,
                    has_down => false,
                    has_left => false,
                    has_right => true)
        port map(  CLK => CLK,
                    P1 => spikes(i - im_length),
                    P2 => '0',
                    P3 => '0',
                    P4 => spikes(i+1),
                    pixel_up => im_value(i - im_length),
                    pixel_down => (others => '0'),
                    pixel_left => (others => '0'),
                    pixel_right => im_value(i+1),
                    pixel_i => im_value(i),
                    G => G,
                    RST => RST,
                    spike_out => spikes(i));
end generate left_down_corner;

right_down_corner: if (i = num_neur) generate
    corner4: Neur_Unit
        generic map(pixel_size => pixel_size,
                    Zi => 2,
                    current_neur => i,
                    num_neur => num_neur,
                    threshold => threshold,
                    has_up => true,
                    has_down => false,
                    has_left => true,
                    has_right => false)
        port map(  CLK => CLK,
                    P1 => spikes(i - im_length),
                    P2 => '0',
                    P3 => spikes(i-1),
                    P4 => '0',
                    pixel_up => im_value(i - im_length),
                    pixel_down => (others => '0'),
                    pixel_left => im_value(i-1),
                    pixel_right => (others => '0'),
```

```
                        pixel_i => im_value(i),
                        G => G,
                        RST => RST,
                        spike_out => spikes(i));
    end generate right_down_corner;

    im_body: if ((i > im_length+1) and (i < (im_width -1)*im_length) and
(i mod im_length /= 0) and (i mod im_length /= 1)) generate
        center: Neur_Unit
            generic map(pixel_size => pixel_size,
                        Zi => 4,
                        current_neur => i,
                        num_neur => num_neur,
                        threshold => threshold,
                        has_up => true,
                        has_down => true,
                        has_left => true,
                        has_right => true)
            port map(   CLK => CLK,
                        P1 => spikes(i - im_length),
                        P2 => spikes(i + im_length),
                        P3 => spikes(i-1),
                        P4 => spikes(i+1),
                        pixel_up => im_value(i - im_length),
                        pixel_down => im_value(i + im_length),
                        pixel_left => im_value(i-1),
                        pixel_right => im_value(i+1),
                        pixel_i => im_value(i),
                        G => G,
                        RST => RST,
                        spike_out => spikes(i));
    end generate im_body;

end generate Neuron;

Glob_Inh: inh
Generic map(    num_neur => num_neur)
Port map(       Spikes_in => spikes,
                G => G);

process(CLK)
begin
    if rising_edge(CLK) then
        Spikes_out <= spikes;
    end if;
end process;

spikes_line <= spikes(80 downto 65);
end Behavioral;
```

1.2.1. Neur_Unit.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 25.03.2018 15:24:51
-- Module Name: TOP - Behavioral
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity Neur_Unit is
generic(pixel_size: integer;
        Zi : integer;
        current_neur: integer;
        num_neur: integer;
        threshold: integer;
        has_up, has_down, has_left, has_right: boolean);
Port ( CLK : in STD_LOGIC;
       P1,P2,P3,P4: in STD_LOGIC;
       pixel_up : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       pixel_down : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       pixel_left : in STD_LOGIC_VECTOR (pixel_size -1  downto 0);
       pixel_right : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       pixel_i : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       RST: in STD_LOGIC;
       G : in STD_LOGIC;
       spike_out: out STD_LOGIC);
end Neur_Unit;

architecture Behavioral of Neur_Unit is

component pix_diff_test is
generic(pixel_size: in integer;
        threshold: in integer;
        has_up, has_down, has_left, has_right: in boolean); -- number of
neighbours
Port ( pixel_up : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       pixel_down : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       pixel_left : in STD_LOGIC_VECTOR (pixel_size -1  downto 0);
       pixel_right : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       pixel_i : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
       num_pass: out STD_LOGIC_VECTOR(2 downto 0);
       alfa : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component lead_detecter is
    Generic(    current_neur: natural);
    Port ( CLK : in STD_LOGIC;
           RST: in STD_LOGIC;
           n_passed : in STD_LOGIC_VECTOR (2 downto 0);
           sum_I_N : out STD_LOGIC_VECTOR (12 downto 0));
end component;

component arithmetic is
    Generic( Zi: integer);
```

```
    Port (  inh : in STD_LOGIC;
            sum_I_N : in STD_LOGIC_VECTOR (12 downto 0);
            P_up : in STD_LOGIC;
            P_down : in STD_LOGIC;
            P_left : in STD_LOGIC;
            P_right : in STD_LOGIC;
            test : in STD_LOGIC_VECTOR (3 downto 0);
            X_n: in STD_LOGIC_VECTOR (11 downto 0);
            X_n1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;

component RAM is
Port ( CLK : in STD_LOGIC;
       X_n1 : in STD_LOGIC_VECTOR (11 downto 0);
       X_n : out STD_LOGIC_VECTOR (11 downto 0));
end component;

-- internal signals --
signal num_pass: STD_LOGIC_VECTOR(2 downto 0);
signal test_aux: std_logic_vector(3 downto 0);
signal X_n, in_RAM, out_RAM: STD_LOGIC_VECTOR (11 downto 0);
signal sum_I_N: STD_LOGIC_VECTOR (12 downto 0);
signal spike_aux: STD_LOGIC;

begin

pdt: pix_diff_test
    Generic map (pixel_size => pixel_size,
                 threshold => threshold,
                 has_up => has_up,
                 has_down => has_down,
                 has_left => has_left,
                 has_right => has_right)
    port map(  pixel_up => pixel_up,
               pixel_down => pixel_down,
               pixel_left => pixel_left,
               pixel_right => pixel_right,
               pixel_i => pixel_i,
               num_pass => num_pass,
               alfa => test_aux);

LD: lead_detecter
    Generic map(current_neur => current_neur)
    port map(   CLK => CLK,
                RST => RST,
                n_passed => num_pass,
                sum_I_N => sum_I_N);

mem_RAM: RAM
    port map (CLK => CLK,
              X_n => out_RAM,
              X_n1 => in_RAM);

arit: arithmetic
    generic map( Zi => Zi)
    port map (  inh => G,
                sum_I_N => sum_I_N,
                P_up => P1,
```

```
                P_down => P2,
                P_left => P3,
                P_right => P4,
                test => test_aux,
                X_n => X_n,
                X_n1 => in_RAM);

with out_RAM select
    X_n <= "000000000000" when "111111111111",
           out_RAM when others;

with out_RAM select
    spike_out <= '1' when "111111111111",
                 '0' when others;

end Behavioral;
```

1.2.2. inh.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 17.04.2018 11:42:06
-- Design Name:
-- Module Name: inh - Behavioral
----------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity inh is
    Generic( num_neur: integer);
    Port ( Spikes_in : in STD_LOGIC_VECTOR (num_neur downto 1);
           G : out STD_LOGIC);
end inh;

architecture Behavioral of inh is

begin

G <= '1' when (Spikes_in > 0) else '0';

end Behavioral;
```

1.2.3. pix_diff_test.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politècnica de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 22.03.2018 15:59:06
-- Module Name: pix_diff_test - Behavioral
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- IMPORTANT NOTE WHEN FIRST SYNTHEZING THESE CODES:
-- If I run Vivado Synthesis on my design, it generates incorrect logic
--  and connects output port to ground with the following warning
message:
-- with warning: [Synth 8-3917] design bja_stat has port stat_event_gnt
driven by constant 0
-- cause: This condition occurs only when you have a set of four or more
compares against
--          a constant and the output of that compare drives a port of
the module.
-- solution: set_param synth.elaboration.rodinMoreOptions
"rt::set_parameter optimizeConstantEq false"

entity pix_diff_test is
    generic(pixel_size: integer;
            threshold: integer;
            has_up, has_down, has_left, has_right: boolean); -- number of
neighbours
    Port (
           pixel_up : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           pixel_down : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           pixel_left : in STD_LOGIC_VECTOR (pixel_size -1  downto 0);
           pixel_right : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           pixel_i : in STD_LOGIC_VECTOR (pixel_size -1 downto 0);
           num_pass: out STD_LOGIC_VECTOR(2 downto 0);
           alfa : out STD_LOGIC_VECTOR (3 downto 0));
end pix_diff_test;

architecture Behavioral of pix_diff_test is

--- aux_signals for the pdt unit
type p_d is array (3 downto 0) of unsigned (pixel_size -1 downto 0);
signal pixel_diff: p_d;

--- signals to count #passed neighbours --
signal n_passed: unsigned(2 downto 0);
type t is array(3 downto 0) of std_logic_vector(2 downto 0);
signal test: t;

begin

--- substraction pixel_j - pixel_i ---
pixel_diff(0) <= (unsigned(pixel_up) - unsigned(pixel_i)) when
(unsigned(pixel_up) > unsigned(pixel_i))
    else (unsigned(pixel_i) - unsigned(pixel_up));
```

```
pixel_diff(1) <= (unsigned(pixel_down) - unsigned(pixel_i)) when
(unsigned(pixel_down) > unsigned(pixel_i))
    else (unsigned(pixel_i) - unsigned(pixel_down));
pixel_diff(2) <= (unsigned(pixel_left) - unsigned(pixel_i)) when
(unsigned(pixel_left) > unsigned(pixel_i))
    else (unsigned(pixel_i) - unsigned(pixel_left));
pixel_diff(3) <=  (unsigned(pixel_right) - unsigned(pixel_i)) when
(unsigned(pixel_right) > unsigned(pixel_i))
    else (unsigned(pixel_i) - unsigned(pixel_right));


--- pixel test, if pixel_diff < 16 ---
test(0) <= "001" when ((pixel_diff(0) < to_unsigned(threshold,
pixel_size)) and has_up) else
           "000";

test(1) <= "001" when ((pixel_diff(1) < to_unsigned(threshold,
pixel_size)) and has_down) else
           "000";

test(2) <= "001" when ((pixel_diff(2) < to_unsigned(threshold,
pixel_size)) and has_left) else
           "000";

test(3) <= "001" when ((pixel_diff(3) < to_unsigned(threshold,
pixel_size)) and has_right) else
           "000";
alfa <= test(3)(0) & test(2)(0) & test(1)(0) & test(0)(0);


--- sum up the vector test to know the #passed neighbours ---
num_pass <= std_logic_vector(unsigned(test(0)) + unsigned(test(1))
                  + unsigned(test(2)) + unsigned(test(3)));


end Behavioral;
```

1.2.4. LFSR.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 19.05.2018 15:34:35
-- Module Name: LFSR - Behavioral
----------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity LFSR is
    Generic(    current_neur: natural);
    Port ( CLK : in STD_LOGIC;
           RST: in STD_LOGIC;
           Noise : out STD_LOGIC_VECTOR(7 downto 0));
end LFSR;

architecture Behavioral of LFSR is

signal noise_aux: std_logic_vector(8 downto 1) :=
STD_LOGIC_VECTOR(to_unsigned(current_neur, 8));
signal xor_term: std_logic;

begin

xor_term <= noise_aux(8) xor noise_aux(6) xor noise_aux(5) xor
noise_aux(4);

process(CLK)
begin
    if rising_edge(CLK) then
            noise_aux <= noise_aux(7) & noise_aux(6) & noise_aux(5) &
noise_aux(4)
                    & noise_aux(3) & noise_aux(2) & noise_aux(1) &
xor_term;
    end if;
end process;

with RST select
Noise <= noise_aux when '0',
        (others=> '0') when others;

end Behavioral;
```

1.2.5. lead_detect.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 23.03.2018 17:13:28
-- Module Name: lead_detecter - Behavioral
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity lead_detecter is
    Generic(    current_neur: natural);
    Port ( CLK : in STD_LOGIC;
           RST: in STD_LOGIC;
           n_passed : in STD_LOGIC_VECTOR (2 downto 0);
           sum_I_N : out STD_LOGIC_VECTOR (12 downto 0));
end lead_detecter;

architecture Behavioral of lead_detecter is
component LFSR is
    Generic(    current_neur: natural);
    Port ( CLK : in STD_LOGIC;
           RST: in STD_LOGIC;
           Noise : out STD_LOGIC_VECTOR(7 downto 0));
end component;

signal leader: boolean;
signal Noise_aux: STD_LOGIC_VECTOR(7 downto 0);
signal I_mod: STD_LOGIC_VECTOR (12 downto 0);
signal sum,sum_shifted: unsigned (12 downto 0);

begin

U1: LFSR
Generic map(    current_neur => current_neur)
port map(       CLK => CLK,
                RST => RST,
                Noise => Noise_aux);

leader <= true when (n_passed = "010" or n_passed = "011") else false;

I_mod <= "1010000000000" when leader
    else (others=>'0') when (n_passed = "000")
    else "0111100110011"; --0.95

sum <= unsigned(I_mod) + unsigned(Noise_aux);
sum_shifted <= shift_right(sum, 5);

process(CLK)
begin
    if rising_edge(CLK) then
        sum_I_N <= std_logic_vector(sum_shifted);
    end if;
end process;
end Behavioral;
```

1.2.6. arithmetic.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 08.03.2018 14:25:22
-- Module Name: arithmetic - Behavioral
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity arithmetic is
    Generic( Zi: integer);
    Port ( inh : in STD_LOGIC;
           sum_I_N : in STD_LOGIC_VECTOR (12 downto 0);
           P_up : in STD_LOGIC;
           P_down : in STD_LOGIC;
           P_left : in STD_LOGIC;
           P_right : in STD_LOGIC;
           test : in STD_LOGIC_VECTOR (3 downto 0);
           X_n: in STD_LOGIC_VECTOR (11 downto 0);
           X_n1 : out STD_LOGIC_VECTOR (11 downto 0));
end arithmetic;

architecture Behavioral of arithmetic is

-- signals for the sum of alfa_ij and P_j
signal S1, S2, S3, S4: std_logic_vector(0 downto 0);
signal sumS12, sumS34: unsigned(1 downto 0):=(others=>'0');
signal sum_P: UNSIGNED (2 downto 0);
signal w_case: STD_LOGIC_VECTOR (2 downto 0);
signal weight_shifted: unsigned (12 downto 0);
signal sum_X_W: unsigned(14 downto 0);
-- signals for the sub part
signal G_shifted : unsigned(11 downto 0);
signal shift_X: unsigned(11 downto 0);
signal sub_X: unsigned(11 downto 0);

-- signals for the final sum
signal pos_term: unsigned(14 downto 0);
signal arithm_sum: signed(14 downto 0);

begin
-----multiplication pj*alfa(i,j)-----
S1(0) <= P_up and test(0);
S2(0) <= P_down and test(1);
S3(0) <= P_left and test(2);
S4(0) <= P_right and test(3);

sumS12 <= '0' & unsigned(S1) + unsigned(S2);
sumS34 <= '0' & unsigned(S3) + unsigned(S4);
sum_P <=  '0' & sumS12 + sumS34;
```

```
--------sum divided by #neighbours-----
weight_shifted <= "0000000000000" when (sum_P = 0) else
-- 0
                  "0100000000000" when (Zi = 2 and sum_P = 1) or (Zi = 4
and sum_P = 2) else       -- 0.5
                  "0010101010101" when (Zi = 3 and sum_P = 1) else
-- 0.33
                  "0101010101011" when (Zi = 3 and sum_P = 2) else
-- 0.66
                  "0010000000000" when (Zi = 4 and sum_P = 1) else
-- 0.25
                  "0110000000000" when (Zi = 4 and sum_P = 3) else
-- 0.75
                  "1000000000000";

---------substractions------------
shift_X <= shift_right(unsigned(X_n),5);
sub_X <= unsigned(X_n) - shift_X;
G_shifted <= "000000110010" when inh = '1' else (others=>'0');
sum_X_W <=   "00" & unsigned(sum_I_N) + sub_X;
pos_term <=  weight_shifted + sum_X_W;
arithm_sum <= signed(pos_term - G_shifted);

--- final sum for X_n1
X_n1 <= (others=> '0') when (arithm_sum(14) = '1') else -- salida cero si
es negativo
        STD_LOGIC_VECTOR(arithm_sum(11 downto 0)) when
((arithm_sum(13)='0') and (arithm_sum(12)='0'))
        else (others => '1');
end Behavioral;
```

1.2.7. RAM.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 27.03.2018 14:12:39
-- Module Name: RAM - Behavioral
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use STD.TEXTIO.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;

entity RAM is
    Port ( CLK : in STD_LOGIC;
           X_n1 : in STD_LOGIC_VECTOR (11 downto 0);
           X_n : out STD_LOGIC_VECTOR (11 downto 0));
end RAM;

architecture Behavioral of RAM is

signal pre_value: std_logic_vector(11 downto 0) := (others=> '0');

begin

process(CLK)
begin
    if (CLK='1' and CLK'event) then
            pre_value <= X_n1;
    end if;
end process;

X_n <= pre_value;

end Behavioral;
```

1.3.Figure_Select.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politècnica de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 07.05.2018 12:17:05
-- Design Name: LEGION
-- Module Name: Figure_select - Behavioral
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use work.bus_multiplexer_pkg.all;

entity Figure_select is
    Generic( num_neur: integer := 192;
             pixel_size: integer := 8);
    Port ( spikes_out : in STD_LOGIC_VECTOR (num_neur downto 1);
           RGB_value: in bus_array(num_neur downto 1, 3 downto 0);
           CLK:  in STD_LOGIC;
           start : in STD_LOGIC;
           up : in STD_LOGIC;
           down : in STD_LOGIC;
           ok : in STD_LOGIC;
           change_img: in STD_LOGIC;
           RGB_display: out bus_array(num_neur downto 1, 3 downto 0);
           SSEG : out STD_LOGIC_VECTOR (7 downto 0);
           CIFRA : out STD_LOGIC_VECTOR (7 downto 0));
end Figure_select;

architecture Behavioral of Figure_select is

component state_machine is
    Port ( CLK : in STD_LOGIC;
           I : in STD_LOGIC;
           O : out STD_LOGIC);
end component;

component D7SEG is
    Port ( CLK : in STD_LOGIC;
           start : in STD_LOGIC;
           up : in STD_LOGIC;
           down : in STD_LOGIC;
           ok : in STD_LOGIC;
           change_img: in STD_LOGIC;
           num_f: in STD_LOGIC_VECTOR(3 downto 0);
           figure_selected: out STD_LOGIC_VECTOR(3 downto 0);
           SSEG : out STD_LOGIC_VECTOR (7 downto 0);
           CIFRA : out STD_LOGIC_VECTOR (7 downto 0));
end component;

--- signals for the state machine
signal start_aux, up_aux, down_aux, ok_aux, rst_aux: std_logic := '0';
type start_botton is (S0, S1);
signal state_s: start_botton := S0;

--- signals to record the figures
type figura is array(9 downto 0) of unsigned(num_neur downto 1);
```

```
signal fig: figura := (others=> (others => '0'));
signal n_found: unsigned(3 downto 0):= "0000";
signal num_f: std_logic_vector(3 downto 0);d
type start_b is(resetting, waiting, recording, stop);
signal rec_state: start_b := resetting;
signal figure_selected: STD_LOGIC_VECTOR(3 downto 0);

-- registers for the output
signal SSEG_aux, CIFRA_aux: std_logic_vector(7 downto 0);
signal sel_fig: std_logic_vector(num_neur downto 1);

-- signals to view the original image
type to_view is (original, figures);
signal view_state: to_view;

begin

-- decoder 7 segments
display7seg: D7SEG
port map(   CLK => CLK,
            start => start_aux,
            up => up_aux,
            down => down_aux,
            ok => ok_aux,
            change_img => rst_aux,
            num_f => num_f,
            figure_selected => figure_selected,
            SSEG => SSEG_aux,
            CIFRA => CIFRA_aux);

-- state machine for bottons up/down/ok
up_botton: state_machine
port map(   CLK => CLK,
            I => up,
            O => up_aux);

down_botton: state_machine
port map(   CLK => CLK,
            I => down,
            O => down_aux);

ok_botton: state_machine
port map(   CLK => CLK,
            I => ok,
            O => ok_aux);

rst_botton: state_machine
port map(   CLK => CLK,
            I => change_img,
            O =>  rst_aux);

-- modified FSM for the start botton, beginning the process with no
spikes on the net
process(CLK)
begin
    if rising_edge(CLK) then
        case state_s is
            when S0 => start_aux <= '0';
```

```
                if (start = '1') then
                    state_s <= S1;
                end if;

            when S1 => start_aux <= '0';
                if (start = '0' and unsigned(spikes_out) = 0) then
                    start_aux <= '1';
                    state_s <= S0;
                end if;
        end case;
    end if;
end process;

-- FSM for the figure searching process
process(CLK)
begin
    if rising_edge(CLK) then
        if (rst_aux = '1') then
            n_found <= "0000";
            rec_STATE <= resetting;
        else
            case rec_state is
                when resetting => n_found <= "0000";
                    if (start_aux = '1') then
                        rec_state <= waiting;
                    end if;

                when waiting =>
                    if (n_found >1) then
                        if fig(to_integer(n_found-1)) = fig(0) then
                            rec_STATE <= STOP;
                        elsif (n_found > 9)  then
                            rec_STATE <= STOP;
                        else
                            if (unsigned(spikes_out) /= 0) then
                                rec_STATE <= recording;
                            end if;
                        end if;
                    else
                        if (unsigned(spikes_out) /= 0) then
                            rec_STATE <= recording;
                        end if;
                    end if;

                when recording =>
                    if (unsigned(spikes_out) = 0) then
                        n_found <= n_found + 1;
                        rec_STATE <= waiting;
                    end if;

                when stop=>

            end case;
        end if;
    end if;
end process;
```

```
-- the actual number of figures found is one less
num_f <= std_logic_vector(n_found - 1) when (n_found > 0) and (n_found <
10)   --number of figures found
    else (others => '0') when n_found = "0000"
    else "1001";

--- recording of the figures
process(CLK)
begin
    if (rising_edge(CLK)) then
        if (rst_aux = '1') then
            fig <= (others => (others=>'0'));
        else
            if (rec_state = waiting) or (rec_state = recording) then
                for i in 1 to num_neur loop
                    if (spikes_out(i) = '1') then
                        fig(to_integer(n_found))(i) <= spikes_out(i);
                    end if;
                end loop;
            end if;
        end if;
    end if;
end process;

-- figure to be displayed
process(CLK)
begin
    if rising_edge(CLK) then
        if (rst_aux = '1') then
            sel_fig <= (others=>'0');
        else
            if (ok_aux = '1') then
                sel_fig <=
std_logic_vector(fig(to_integer(unsigned(figure_selected))));
            end if;
        end if;
    end if;
end process;


-- image to be viewed
process(CLK)
begin
    if rising_edge(CLK) then
        case view_state is
            when original =>
                RGB_display <= RGB_value;
                if (ok_aux = '1') then
                    view_state <= figures;
                end if;

            when figures =>
                for i in 1 to num_neur loop
                    for j in 0 to 3 loop
                        RGB_display(i,j) <= not sel_fig(i);
                    end loop;
                end loop;
                if (rst_aux = '1') then
```

```
                    view_state <= original;
                end if;
        end case;
    end if;
end process;

-- output registers
process(CLK)
begin
    if rising_edge(CLK) then
        SSEG <= SSEG_aux;
        CIFRA <= CIFRA_aux;
    end if;

end process;

end Behavioral;
```

1.3.1. D7SEG.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 08.05.2018 16:23:25
-- Module Name: D7SEG - Behavioral
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity D7SEG is
    Port ( CLK : in STD_LOGIC;
           start : in STD_LOGIC;
           up : in STD_LOGIC;
           down : in STD_LOGIC;
           ok : in STD_LOGIC;
           change_img: in STD_LOGIC;
           num_f: in STD_LOGIC_VECTOR(3 downto 0);
           figure_selected: out STD_LOGIC_VECTOR(3 downto 0);
           SSEG : out STD_LOGIC_VECTOR (7 downto 0);
           CIFRA : out STD_LOGIC_VECTOR (7 downto 0));
end D7SEG;

architecture Behavioral of D7SEG is

component prescaler_display is
    Port ( CLK : in STD_LOGIC;
           O : out STD_LOGIC);
end component;

--- signals for the prescaler
signal c: std_logic;
signal counter: std_logic_vector(2 downto 0) := "000";

--- signals for the display
signal DATA: STD_LOGIC_VECTOR (4 downto 0);
signal letter_5,letter_4,letter_3,letter_2,letter_1: STD_LOGIC_VECTOR (3
downto 0);
signal actual_value: STD_LOGIC_VECTOR (4 downto 0) :="00000";
type display is (dot, found, sele, outx);
signal ESTADO: display := dot;

begin

prescaler: prescaler_display
port map(   CLK => CLK,
            O => c);

-- prescaler
process(CLK)
begin
    if rising_edge(CLK) then
        if (c = '1') then
            counter <= counter + 1;
        end if;
    end if;
```

```
end process;

--Decoder 7SEG General
with DATA select
    SSEG    <=  "00000011" when "00000",        --0
                "10011111" when "00001",        --1
                "00100101" when "00010",        --2
                "00001101" when "00011",        --3
                "10011001" when "00100",        --4
                "01001001" when "00101",        --5 / S
                "01000001" when "00110",        --6
                "00011111" when "00111",        --7
                "00000001" when "01000",        --8
                "00001001" when "01001",        --9
                "11111101" when "10000",        --dash
                "01110001" when "10001",        --F
                "10000011" when "10010",        --U
                "10000101" when "10011",        --D
                "11100001" when "10100",        --t
                "01100001" when "10101",        --E
                "11100011" when "10110",        --L
                "11010101" when "10111",        --n
                "00000011" when "11000",        --O
                "01001001" when "11001",        --S
                "11111110" when "11111",        --DOT
                "11111111" when others;         --nothing

--selection of the digit to be displayed
with COUNTER select
    CIFRA <= "11111110" when "000",     --1ª cifra: el # de figura
seleccionado
             "11111101" when "001",     --letras de notificación
             "11111011" when "010",
             "11110111" when "011",
             "11101111" when "100",
             "11011111" when "101",
             "10111111" when "110",
             "01111111" when others;

--selection of the data to be displayed in each digit
 with COUNTER select
     DATA <= actual_value when "000",  --number of found figures or of
the figure selected
             ('1' & letter_5) when "011",
             ('1' & letter_4) when "100",
             ('1' & letter_3) when "101",
             ('1' & letter_2) when "110",
             ('1' & letter_1) when "111",
             "11111" when others;       --dot, before any botton was
pressed

--definition of all possible values of the signals letter_4-4 according
to the state
process(CLK)
begin
if rising_edge(CLK) then
    if (change_img = '1') then
        ESTADO <= dot;
```

```
    else
    case ESTADO is
        when dot => --all dots
            letter_5 <= "1111";
            letter_4 <= "1111";
            letter_3 <= "1111";
            letter_2 <= "1111";
            letter_1 <= "1111";
            actual_value <= "11111";
            if (start = '1') then
                ESTADO <= found;
            end if;

        when found =>
            letter_5 <= "0011";  --D
            letter_4 <= "0111";  --n
            letter_3 <= "0010";  --U
            letter_2 <= "1000";  --O
            letter_1 <= "0001";  --F
            actual_value <= '0' & num_f; -- number of figures found
            if (up = '1') or (down = '1') then
                ESTADO <= sele;
            end if;

        when sele =>
            letter_5 <= "1111";  --dot
            letter_4 <= "0101";  --E
            letter_3 <= "0110";  --L
            letter_2 <= "0101";  --E
            letter_1 <= "1001";  --S
            if (ok = '1') then
                ESTADO <= outx;
            else
                if(up = '1') and (actual_value < num_f) then
                    actual_value <= actual_value + 1;
                elsif (down = '1') and (actual_value > 0) then
                    actual_value <= actual_value - 1;
                end if;
            end if;

        when outx =>
            letter_5 <= "1111";  --dot
            letter_4 <= "1111";  --dot
            letter_3 <= "0100";  --T
            letter_2 <= "0010";  --U
            letter_1 <= "1000";  --O
            if (up = '1') or (down = '1') then
            ESTADO <= sele;
            end if;
    end case;
    end if;
end if;
end process;

figure_selected <= actual_value(3 downto 0);
end Behavioral;
```

1.3.2. prescaler_display.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 09.05.2018 20:15:44
-- Design Name:
-- Module Name: prescaler_display - Behavioral
----------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity prescaler_display is
    Port ( CLK : in STD_LOGIC;
           O : out STD_LOGIC);
end prescaler_display;

architecture Behavioral of prescaler_display is
signal c: STD_LOGIC_VECTOR(16 downto 0) := (others => '0');

begin

-- defition of the counter, speed of the display = 1 kHz
process(CLK)
begin
    if rising_edge(CLK) then
        c <= c + 1;
    end if;
end process;

with c select
    O <= '1' when ("11111111111111111"),
         '0' when others;

end Behavioral;
```

1.3.3. state_machine.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 07.05.2018 15:17:07
-- Module Name: state_machine - Behavioral
----------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity state_machine is
    Port ( CLK : in STD_LOGIC;
           I : in STD_LOGIC;
           O : out STD_LOGIC);
end state_machine;

architecture Behavioral of state_machine is
type one_push is (S0, S1, S2);
signal state_3b: one_push;

begin

process(CLK)
begin
    if rising_edge(CLK) then
        case state_3b is
            when S0 => O <= '0';
                if (I = '1') then
                    state_3b <= S1;
                end if;

            when S1 => O <= '1';
                state_3b <= S2;

            when S2 => O <= '0';
                if (I = '0') then
                    state_3b <= S0;
                end if;
        end case;
    end if;
end process;

end Behavioral;
```

1.4.display_screen.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 10.05.2018 12:47:07
-- Design Name:
-- Module Name: display_screen - Behavioral
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.bus_multiplexer_pkg.all;

entity display_screen is
    Generic(    im_length: integer := 16;
                im_width: integer := 12;
                num_neur : integer := 192);
    Port ( CLK : in STD_LOGIC;
           DATA_IN : in bus_array(num_neur downto 1, 3 downto 0);
           DATA_OUT : out STD_LOGIC_VECTOR (11 downto 0);
           HS : out STD_LOGIC;
           VS : out STD_LOGIC);
end display_screen;

architecture Behavioral of display_screen is

type img is array (num_neur downto 1) of std_logic_vector(3 downto 0);
signal img_in: img;
-- signals for the display
signal pre_mod4:  std_logic_vector(1 downto 0) := "00";
signal pre_h: std_logic_vector(9 downto 0) := (others => '0');
signal pre_v: std_logic_vector(9 downto 0) := (others => '0');
signal transition, pos_img: std_logic;
-- signals to resize the image
signal h40: natural range 144 to 783 := 144;
signal v40: natural range 32 to 511 := 32;
signal i: natural range 1 to num_neur := 1;
signal X: natural range 1 to im_length:= 1;
signal Y: natural range 0 to im_width:= 0;
begin

conv_to_array:
for i in 1 to num_neur generate
    Gray_value: for j in 0 to 3 generate
                    img_in(i)(j) <= DATA_IN(i,j);
                end generate;
end generate;

-- prescaler module 4, f = 25 MHz
process(CLK)
begin
    if rising_edge(CLK) then
            pre_mod4 <= pre_mod4 + 1;
    end if;
end process;
```

```
-- prescaler for an horizontal line
process(CLK)
begin
    if rising_edge(CLK) then
        if (pre_mod4 = "11") then
            if (pre_h = 799) then
                pre_h <= (others => '0');
            else
                pre_h <= pre_h + 1;
            end if;
        end if;
    end if;
end process;

-- prescaler for the vertical
process(CLK)
begin
    if rising_edge(CLK) then
        if (pre_h = 799) and (pre_mod4 = "11") then
            if (pre_v = 520) then
                pre_v <= (others=> '0');
            else
                pre_v <= pre_v + 1;
            end if;
        end if;
    end if;
end process;

transition <= '1' when ((pre_h <144) or (pre_h>783) or (pre_v <31) or
(pre_v >510)) else '0';

--adapting the image to the resolution of the screen, by increasing it 40
times in both length and width
process(CLK)
begin
if rising_edge(CLK) then
    if (transition = '0') then
        if (pre_h = (h40 + 40)) then
            X <= X + 1;
            h40 <= h40 + 40;
        end if;
    else
        h40 <= 144;
        X <= 1;
        if (pre_v > 31) then
            if (pre_v = (v40 + 40)) then
                v40 <= v40 + 40;
                Y <= Y + 1;
            end if;
        else
            v40 <= 32;
            Y <= 0;
        end if;
    end if;
end if;
end process;

i <= X + im_length*Y;
```

```
process(CLK)
begin
    if rising_edge(CLK) then
        if (pre_h <96) then
            HS <= '0';
        else
            HS <= '1';
        end if;

        if (pre_v < 2) then
            VS <= '0';
        else
            VS  <= '1';
        end if;

        if (transition = '1') then
            data_out <= (others => '0');
        else
            DATA_OUT(3 downto 0) <= img_in(i);
            DATA_OUT(7 downto 4) <= img_in(i);
            DATA_OUT(11 downto 8) <= img_in(i);
        end if;
    end if;

end process;

end Behavioral;
```

1.5.input_img.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 18.05.2018 13:58:14
-- Module Name: inpu_img - Behavioral
----------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package bus_multiplexer_pkg is
        type bus_array is array(natural range <>, natural range <>) of
std_logic;

        procedure slm_row_from_slv(signal slm : out bus_array; constant
row : natural; signal slv : in std_logic_vector);
        procedure slv_from_slm_row(signal slv : out std_logic_vector;
signal slm : in bus_array; constant row : natural);
end package;

package body bus_multiplexer_pkg is
        procedure slm_row_from_slv(signal slm : out bus_array; constant
row : natural; signal slv : in std_logic_vector) is
        begin
                for i in slv'range loop
                        slm(row, i) <= slv(i);
                end loop;
        end procedure;

        procedure slv_from_slm_row(signal slv : out std_logic_vector;
signal slm : in bus_array; constant row : natural) is
        begin
                for i in slv'range loop
                        slv(i) <= slm(row, i);
                end loop;
        end procedure;
end package body;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.bus_multiplexer_pkg.all;

entity input_img is
    Generic(    num_neur: natural := 192;
                pixel_size: natural := 8);
    Port ( sel_img : in STD_LOGIC_VECTOR (14 downto 0);
           img_value : out bus_array(num_neur downto 1, pixel_size-1
downto 0);
           RGB_value: out bus_array(num_neur downto 1, 3 downto 0));
end input_img;

architecture Behavioral of input_img is


type im_aux is array (1 to num_neur) of natural range 0 to 255;
```

```
signal img1_aux, img2_aux, img3_aux, img4_aux, img5_aux, img6_aux,
img7_aux, img_rst_aux : im_aux;

type im is array (num_neur downto 1) of std_logic_vector(pixel_size -1
downto 0);
signal im_value: im;
signal img1, img2, img3, img4, img5, img6, img7, img_rst: im;

type im_shift is array (num_neur downto 1) of std_logic_vector(3 downto
0);
signal im_RGB: im_shift;

begin

-- cuadrados
img1_aux <=
(250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  111,  111,  111,  111,  111,  111,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  111,  111,  111,  111,  111,  111,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  111,  111,  111,  111,  111,  111,  250,  250,  250,  50,  50,
50,  50,  250,  250,
 250,  111,  111,  111,  111,  111,  111,  250,  250,  250,  50,  50,
50,  50,  250,  250,
 250,  111,  111,  111,  111,  111,  111,  250,  250,  250,  50,  50,
50,  50,  250,  250,
 250,  111,  111,  111,  111,  111,  111,  250,  250,  250,  50,  50,
50,  50,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250);

-- puente
img2_aux  <=
(250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,   10,   10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,
10,  250,
 250,   10,   10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,
10,  250,
 250,   10,   10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,
10,  250,
 250,  250,  150,  150,  250,  250,  250,  250,  250,  250,  250,  250,
150,  150,  250,  250,
 250,  250,  150,  150,  250,  250,  250,  250,  250,  250,  250,  250,
150,  150,  250,  250,
```

```
 250,  250,  150,  150,  250,  250, 250, 250, 250, 250, 250, 250, 150,
150,  250,  250,
 250,  250,  150,  150,  250,  250, 250, 250, 250, 250, 250, 250, 150,
150,  250,  250,
 250,  250,  250,  250,  250,  250, 250, 250, 250, 250, 250, 250, 250,
250,  250,  250,
 250,  250,  250,  250,  250,  250, 250, 250, 250, 250, 250, 250, 250,
250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250);

-- numeros 23
 img3_aux <=
 (250,  250,   250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
  250,  10,  10,   10,  10,  10,  10,  250,  250,  10,  10,  10,  10,
10,  10,  250,
  250,  10,   10,  10,  10,  10,  10,  250,  250,  10,  10,  10,  10,
10,  10,  250,
  250,  250,  250,   250,  250,  10,  10,  250,  250,  250,  250,  250,
250,  10,  10,  250,
  250,  250,   250,  250,  250,  10,  10,  250,  250,  250,  250,  250,
250,  10,  10,  250,
  250,  10,  10,   10,  10,  10,  10,  250,  250,  10,  10,  10,  10,
10,  10,  250,
  250,  10,   10,  10,  10,  10,  10,  250,  250,  10,  10,  10,  10,
10,  10,  250,
  250,  10,  10,   250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  10,  10,  250,
  250,  10,   10,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  10,  10,  250,
  250,  10,  10,   10,  10,  10,  10,  250,  250,  10,  10,  10,  10,
10,  10,  250,
  250,  10,   10,  10,  10,  10,  10,  250,  250,  10,  10,  10,  10,
10,  10,  250,
  250,  250,  250,   250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250);

--image2: casa
img4_aux <=
(250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  150,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,   250,   250,   150,  150,  150,  250,  250,  10,
10,  250,  250,  250,
 250,  250,  250,  250,   250,  150,   150,  150,  150,  150,  250,  10,
10,  250,  250,  250,
 250,  250,  250,  250,  150,  150,  150,  150,  150,  150,  150,  10,
10,  250,  250,  250,
 250,  250,  250,  150,  150,  150,  150,  150,  150,  150,  150,  150,
10,  250,  250,  250,
 250,  250,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
150,  250,  250,  250,
 250,  250,  250,  50,  50,  50,  50,  50,  50,  50,  50,  250,
250,  250,  250,
 250,  250,  250,  50,  50,  50,  50,  50,  50,  50,  50,  250,
250,  250,  250,
```

```
 250,  250,  250,  50,  50,  50,  50,  50,  50,  50,  50,  250,
250,  250,  250,
 250,  250,  250,  50,  50,  50,  50,  50,  50,  50,  50,  250,
250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250);

img5_aux <= --rectangulos dentro de rectangulos
 (250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
  250,  250,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
150,  150,  250,  250,
  250,  250,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
150,  150,  250,  250,
  250,  250,  150,  150,  250,  250,  250,  250,  250,  250,  250,  250,
150,  150,  250,  250,
  250,  250,  150,  150,  250,  150,  150,  150,  150,  150,  150,  250,
150,  150,  250,  250,
  250,  250,  150,  150,  250,  150,  150,  150,  150,  150,  150,  250,
150,  150,  250,  250,
  250,  250,  150,  150,  250,  250,  250,  250,  250,  250,  250,  250,
150,  150,  250,  250,
  250,  250,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
150,  150,  250,  250,
  250,  250,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
150,  150,  250,  250,
  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250);

-- image5: happy face
img6_aux <=
(250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  10,  10,  10,  250,  250,  250,  250,  10,  10,  10,
250,  250,  250,
 250,  250,  250,  10,  10,  10,  250,  250,  250,  250,  10,  10,  10,
250,  250,  250,
 250,  250,  250,  10,  10,  10,  250,  250,  250,  250,  10,  10,  10,
250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250,
 250,  250,  250,  10,  10,  250,  250,  250,  250,  250,  250,  10,  10,
250,  250,  250,
 250,  250,  250,  10,  10,  10,  250,  250,  250,  250,  10,  10,  10,
250,  250,  250,
 250,  250,  250,  250,  10,  10,  10,  10,  10,  10,  10,  10,  250,
250,  250,  250,
 250,  250,  250,  250,  250,  10,  10,  10,  10,  10,  10,  250,  250,
250,  250,  250,
```

```
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250,
250,  250,  250,  250);

-- image6: espiral
img7_aux <=
 (250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250, 250,
250, 250,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250, 250,
250,  50,  250,  250,
 250,  250,  250,   50,   50,   50,   50,   50,   50,   50,   50,  50,
250,  50,  250,  250,
 250,  250,  250,   50,  250,  250,  250,  250,  250,  250,  250,  50,
250,  50,  250,  250,
 250,  250,  250,   50,  250,   50,   50,   50,   50,   50,  250,  50,
250,  50,  250,  250,
 250,  250,  250,   50,  250,   50,  250,  250,  250,   50,  250,  50,
250,  50,  250,  250,
 250,  250,  250,   50,  250,   50,  250,   50,   50,   50,  250,  50,
250,  50,  250,  250,
 250,  250,  250,   50,  250,   50,  250,  250,  250,  250,  250,  50,
250,  50,  250,  250,
 250,  250,  250,   50,  250,   50,   50,   50,   50,   50,   50,  50,
250,  50,  250,  250,
 250,  250,  250,   50,  250,  250,  250,  250,  250,  250,  250, 250,
250,  50,  250,  250,
 250,  250,  250,   50,   50,   50,   50,   50,   50,   50,   50,  50,
50,  50,  250,  250,
 250,  250,  250,  250,  250,  250,  250,  250,  250,  250,  250, 250,
250, 250,  250,  250);


img_rst_aux <=
(250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,
70,  250,  70,
 70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,
250,  70,  250,
 250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,
70,  250,  70,
 70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,
250,  70,  250,
 250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,
70,  250,  70,
 70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,
250,  70,  250,
 250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,
70,  250,  70,
 70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,
250,  70,  250,
 250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,
70,  250,  70,
 70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,
250,  70,  250,
 250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,
70,  250,  70,
 70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,  250,  70,
250,  70,  250);

image_gener:
```

```
for j in 1 to num_neur generate
    img1(j) <= std_logic_vector(to_unsigned(img1_aux(j), pixel_size));
    img2(j) <= std_logic_vector(to_unsigned(img2_aux(j), pixel_size));
    img3(j) <= std_logic_vector(to_unsigned(img3_aux(j), pixel_size));
    img4(j) <= std_logic_vector(to_unsigned(img4_aux(j), pixel_size));
    img5(j) <= std_logic_vector(to_unsigned(img5_aux(j), pixel_size));
    img6(j) <= std_logic_vector(to_unsigned(img6_aux(j), pixel_size));
    img7(j) <= std_logic_vector(to_unsigned(img7_aux(j), pixel_size));
    img_rst(j) <= std_logic_vector(to_unsigned(img_rst_aux(j),
pixel_size));
end generate image_gener;

with sel_img select
    im_value <= img1 when "000000000000001",
                img2 when "000000000000010",
                img3 when "000000000000100",
                img4 when "000000000001000",
                img5 when "000000000010000",
                img6 when "000000000100000",
                img7 when "000000001000000",
                img_rst when others;

gray_to_RGB:
for i in 1 to num_neur generate
    im_RGB(i) <= im_value(i)(7) & im_value(i)(6) & im_value(i)(5) &
im_value(i)(4);
    pixel_to_pixel:
        for b in 0 to 3 generate
            RGB_value(i,b) <= im_RGB(i)(b);
        end generate pixel_to_pixel;
end generate;

uut1: for j in 1 to num_neur generate
    uut2: for k in 0 to pixel_size-1 generate
        img_value(j,k) <= im_value(j)(k);
    end generate uut2;
end generate uut1;

end Behavioral;
```

2. simulation sources

2.1.TOP_TB.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 10.05.2018 10:40:20
-- Module Name: TOP_TESTBENCH - Behavioral
----------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity TOP_TESTBENCH is
Generic (  threshold: in integer:= 16;
           num_neur: integer := 192;
           pixel_size: integer := 8;
           im_length: integer := 16;
           im_width: integer := 12);
end TOP_TESTBENCH;

architecture Behavioral of TOP_TESTBENCH is

component TOP is
Generic (  threshold: in integer;
           num_neur: integer;
           pixel_size: integer;
           im_length: integer;
           im_width: integer);

Port (  CLK: in STD_LOGIC;
        start : in STD_LOGIC;
        up : in STD_LOGIC;
        down : in STD_LOGIC;
        ok : in STD_LOGIC;
        change_img: in STD_LOGIC;
        sel_img: in std_logic_vector(14 downto 0);
        DATA_OUT : out STD_LOGIC_VECTOR (11 downto 0);
        HS : out STD_LOGIC;
        VS : out STD_LOGIC;
        SSEG : out STD_LOGIC_VECTOR (7 downto 0);
        CIFRA : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal CLK,start,up,down,ok,rst: std_logic;
signal SSEG, CIFRA: STD_LOGIC_VECTOR(7 downto 0);
signal sel_img: std_logic_vector(14 downto 0);
signal HS, VS: STD_LOGIC;
signal DATA_OUT: STD_LOGIC_VECTOR(11 downto 0);


constant CLK_period : time := 10 ns;

begin
```

```
UUT: TOP
generic map(    pixel_size => pixel_size,
                num_neur => num_neur,
                threshold => threshold,
                im_length => im_length,
                im_width => im_width)
port map(   CLK => CLK,
            start => start,
            up => up,
            down => down,
            ok => ok,
            change_img => rst,
            SSEG => SSEG,
            sel_img => sel_img,
            CIFRA => CIFRA,
            DATA_OUT => DATA_OUT,
            HS => HS,
            VS => VS);

-- Clock process definitions
CLK_process :process
begin
    CLK <= '0';
    wait for CLK_period/2;
    CLK <= '1';
    wait for CLK_period/2;
end process;

stim_process: process
begin
    sel_img <= "000000000000001";
    rst <= '0';
    start <= '0';
    up <= '0';
    down <= '0';
    ok <= '0';
    wait for CLK_period/2;

    wait for 5 ms;

    start <= '1';
    wait for 30 us;
    start <= '0';
    wait for 50 us;

    up <= '1';
    wait for 30 us;
    up <= '0';
    wait for 35 us;
    up <= '1';
    wait for 30 us;
    up <= '0';
    wait for 35 us;

    ok <= '1';
    wait for 30 us;
    ok <= '0';
    wait for 30 us;
```

```
        down <= '1';
        wait for 30 us;
        down <= '0';
        wait for 35 us;
        ok <= '1';
        wait for 30 us;
        ok <= '0';
        wait for 30 us;

        rst <= '1';
        sel_img <= "000000000000111";
        wait for 10 us;

        rst <= '0';
        sel_img <= "000000000000010";
        wait for 5 ms;
        -- to see the simulation of the behaviour of the spikes and the
selection process copy the previous code here
        wait;
end process;
end Behavioral;
```

2.2.net_TB.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 18.04.2018 16:47:53
-- Module Name: net_TB - Behavioral
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use work.bus_multiplexer_pkg.all;

entity net_TB is
    Generic ( threshold: integer:= 16;
              pixel_size: integer := 8;
              num_neur: integer := 192;
              im_length: integer := 16;
              im_width: integer := 12);
end net_TB;

architecture Behavioral of net_TB is

component Neur_Net is
    Generic (  threshold: integer;
          num_neur: integer;
          pixel_size: integer;
          im_length: integer;
          im_width: integer);
Port (  CLK: in STD_LOGIC;
        RST: in STD_LOGIC;
        img_value : in bus_array(num_neur downto 1, pixel_size-1 downto
0);
        spikes_line: out STD_LOGIC_VECTOR(15 downto 0);
        Spikes_out : out STD_LOGIC_VECTOR (num_neur downto 1));
end component;

component input_img is
    Generic(    num_neur: natural;
             pixel_size: natural);
Port ( sel_img : in STD_LOGIC_VECTOR (14 downto 0);
       img_value : out bus_array(num_neur downto 1, pixel_size-1 downto
0);
       RGB_value: out bus_array(num_neur downto 1, 3 downto 0));
end component;

signal CLK, RST: STD_LOGIC;
signal Spikes : STD_LOGIC_VECTOR (num_neur downto 1);
signal sel_img: std_logic_vector(14 downto 0);
signal spikes_line: STD_LOGIC_VECTOR(15 downto 0);

signal im_aux: bus_array(num_neur downto 1, pixel_size-1 downto 0);
signal RGB_aux, RGB_out:bus_array(num_neur downto 1, 3 downto 0);

constant CLK_period : time := 10 ns;

---input
```

```
begin
UUT1: Neur_Net
    generic map(     threshold => threshold,
                     num_neur => num_neur,
                     pixel_size => pixel_size,
                     im_length => im_length,
                     im_width => im_width)
port map(        CLK => CLK,
                 img_value => im_aux,
                 spikes_line => spikes_line,
                 RST => RST,
                 spikes_out => spikes);

UUT2: input_img
    generic map( num_neur => num_neur,
                 pixel_size => pixel_size)
    port map(       sel_img => sel_img,
                    img_value => im_aux,
                    RGB_value => RGB_aux);

-- Clock process definitions
CLK_process :process
begin
    CLK <= '0';
    wait for CLK_period/2;
    CLK <= '1';
    wait for CLK_period/2;
end process;


stimu_process: process
begin
    sel_img <= "000010000000001";
    RST <= '0';
    wait for CLK_period/2;
    wait for 500 us;

    RST <= '1';
    sel_img <= "000000000001110";
    wait for 100 us;

    RST <= '0';
    sel_img <= "000000000000010";
    wait for 500 us;

    RST <= '1';
    sel_img <= "000000000001110";
    wait for 100 us;

    RST <= '0';
    sel_img <= "000000000000100";
    wait for 500 us;

    RST <= '1';
    sel_img <= "000000000001110";
    wait for 100 us;

    RST <= '0';
```

```
    sel_img <= "0000000000001000";
    wait for 500 us;

    RST <= '1';
    sel_img <= "0000000000001110";
    wait for 100 us;

    RST <= '0';
    sel_img <= "0000000000010000";
    wait for 500 us;

    RST <= '1';
    sel_img <= "0000000000001110";
    wait for 100 us;

    RST <= '0';
    sel_img <= "0000000000100000";
    wait for 500 us;

    RST <= '1';
    sel_img <= "0000000000001110";
    wait for 100 us;

    RST <= '0';
    sel_img <= "0000000001000000";
    wait for 500 us;

    wait;
end process;

end Behavioral;
```

2.3.control_TB.vhd

```
----------------------------------------------------------------------------
-- University: Universitat Politécnic de Catalunya
-- Engineer: Xinqiu Ye
-- Create Date: 07.05.2018 14:49:41
-- Module Name: control_tb - Behavioral
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.bus_multiplexer_pkg.all;

entity control_tb is
    Generic( num_neur: integer := 192;
             pixel_size: integer := 8);
end control_tb;

architecture Behavioral of control_tb is

component Figure_select is
Generic( num_neur: integer := 192;
         pixel_size: integer := 8);
Port ( spikes_out : in STD_LOGIC_VECTOR (num_neur downto 1);
       RGB_value: in bus_array(num_neur downto 1, 3 downto 0);
       CLK:  in STD_LOGIC;
       start : in STD_LOGIC;
       up : in STD_LOGIC;
       down : in STD_LOGIC;
       ok : in STD_LOGIC;
       change_img: in STD_LOGIC;
       RGB_display: out bus_array(num_neur downto 1, 3 downto 0);
       SSEG : out STD_LOGIC_VECTOR (7 downto 0);
       CIFRA : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal CLK, start, up, down, ok, rst: std_logic;
signal sel_fig, spikes_out: STD_LOGIC_VECTOR (num_neur downto 1) :=
(others=> '0');
signal SSEG, CIFRA: STD_LOGIC_VECTOR (7 downto 0);
signal RGB_value, RGB_display: bus_array(num_neur downto 1, 3 downto 0);
constant CLK_period: time := ns;
begin

UUT: Figure_select
generic map (num_neur => num_neur,
             pixel_size => pixel_size)
port map (  CLK => CLK,
            start => start,
            up => up,
            down => down,
            ok => ok,
            change_img => rst,
            spikes_out => spikes_out,
            SSEG => SSEG,
            CIFRA => CIFRA,
            RGB_display => RGB_display,
            RGB_value => RGB_value);
```

```
CLK_process: process
begin
    CLK <= '1';
    wait for CLK_period/2;
    CLK <= '0';
    wait for CLK_period/2;
end process;

stim_process: process
begin
    spikes_out <= (others=>'0');
    rst <= '0';
    start <= '0';
    up <= '0';
    down <= '0';
    ok <= '0';
    wait for 1135 ns;
    wait for 5 us;

    start <= '1';
    wait for 2 us;
    start <= '0';
    wait for 10 us;

    spikes_out(1) <= '1';
    spikes_out(2) <= '1';
    spikes_out(3) <= '1';
    spikes_out(4) <= '1';
    wait for CLK_period;

    spikes_out(1) <= '0';
    spikes_out(2) <= '0';
    spikes_out(3) <= '0';
    spikes_out(4) <= '0';
    wait for CLK_period;

    wait for 5 us;

    spikes_out(192) <= '1';
    spikes_out(191) <= '1';
    spikes_out(190) <= '1';
    spikes_out(189) <= '1';
    wait for CLK_period;

    spikes_out(192) <= '0';
    spikes_out(191) <= '0';
    spikes_out(190) <= '0';
    spikes_out(189) <= '0';
    wait for CLK_period;
    wait for 5 us;

    spikes_out(1) <= '1';
    spikes_out(2) <= '1';
    spikes_out(3) <= '1';
    spikes_out(4) <= '1';
    wait for CLK_period;
```

```
spikes_out(1) <= '0';
spikes_out(2) <= '0';
spikes_out(3) <= '0';
spikes_out(4) <= '0';
wait for CLK_period;

wait for 5 us;

spikes_out(192) <= '1';
spikes_out(191) <= '1';
spikes_out(190) <= '1';
spikes_out(189) <= '1';
wait for CLK_period;

spikes_out(192) <= '0';
spikes_out(191) <= '0';
spikes_out(190) <= '0';
spikes_out(189) <= '0';
wait for CLK_period;
wait for 5 us;

spikes_out(1) <= '1';
spikes_out(2) <= '1';
spikes_out(3) <= '1';
spikes_out(4) <= '1';
wait for CLK_period;

spikes_out(1) <= '0';
spikes_out(2) <= '0';
spikes_out(3) <= '0';
spikes_out(4) <= '0';
wait for CLK_period;

wait for 5 us;

spikes_out(192) <= '1';
spikes_out(191) <= '1';
spikes_out(190) <= '1';
spikes_out(189) <= '1';
wait for CLK_period;

spikes_out(192) <= '0';
spikes_out(191) <= '0';
spikes_out(190) <= '0';
spikes_out(189) <= '0';
wait for CLK_period;
wait for 5 us;

up <= '1';
wait for 10 us;
up <= '0';
wait for 15 us;
down <= '1';
wait for 10 us;
down <= '0';
wait for 10 us;
ok <= '1';
wait for 30 us;
```

```
    ok <= '0';
    wait for 10 us;

    rst <= '1';
    wait for 10 us;
    rst <= '0';
    wait for 20 us;

    start <= '1';
    wait for 2 us;
    start <= '0';
    wait for 100 us;

    up <= '1';
    wait for 30 us;
    up <= '0';
    wait for 35 us;
    ok <= '1';
    wait for 30 us;
    ok <= '0';
    wait;
end process;


end Behavioral;
```

3. XDC contraints - constraints_TOP.xdc

```
create_clock -period 10.000 -name CLK -waveform {0.000 5.000} [get_ports
CLK]
set_input_delay -clock [get_clocks CLK] -min -add_delay 1.000 [get_ports
down]
set_input_delay -clock [get_clocks CLK] -max -add_delay 2.000 [get_ports
down]
set_input_delay -clock [get_clocks CLK] -min -add_delay 1.000 [get_ports
ok]
set_input_delay -clock [get_clocks CLK] -max -add_delay 2.000 [get_ports
ok]
set_input_delay -clock [get_clocks CLK] -min -add_delay 1.000 [get_ports
change_img]
set_input_delay -clock [get_clocks CLK] -max -add_delay 2.000 [get_ports
change_img]
set_input_delay -clock [get_clocks CLK] -min -add_delay 1.500 [get_ports
start]
set_input_delay -clock [get_clocks CLK] -max -add_delay 2.500 [get_ports
start]
set_input_delay -clock [get_clocks CLK] -min -add_delay 1.000 [get_ports
up]
set_input_delay -clock [get_clocks CLK] -max -add_delay 2.000 [get_ports
up]
set_input_delay -clock [get_clocks CLK] -min -add_delay 1.000 [get_ports
RST]
set_input_delay -clock [get_clocks CLK] -max -add_delay 2.000 [get_ports
RST]
set_input_delay -clock [get_clocks CLK] -min -add_delay 0.005 [get_ports
{sel_img[*]}]
set_input_delay -clock [get_clocks CLK] -max -add_delay 0.050 [get_ports
{sel_img[*]}]
set_output_delay -clock [get_clocks CLK] -min -add_delay 0.200 [get_ports
{CIFRA[*]}]
set_output_delay -clock [get_clocks CLK] -max -add_delay 0.700 [get_ports
{CIFRA[*]}]
set_output_delay -clock [get_clocks CLK] -min -add_delay 0.200 [get_ports
{SSEG[*]}]
set_output_delay -clock [get_clocks CLK] -max -add_delay 0.700 [get_ports
{SSEG[*]}]
set_output_delay -clock [get_clocks CLK] -min -add_delay 0.200 [get_ports
{spikes_line[*]}]
set_output_delay -clock [get_clocks CLK] -max -add_delay 0.500 [get_ports
{spikes_line[*]}]
set_output_delay -clock [get_clocks CLK] -min -add_delay 0.005 [get_ports
{DATA_OUT[*]}]
set_output_delay -clock [get_clocks CLK] -max -add_delay 0.010 [get_ports
{DATA_OUT[*]}]
set_output_delay -clock [get_clocks CLK] -min -add_delay 0.005 [get_ports
HS]
set_output_delay -clock [get_clocks CLK] -max -add_delay 0.010 [get_ports
HS]
set_output_delay -clock [get_clocks CLK] -min -add_delay 0.005 [get_ports
VS]
set_output_delay -clock [get_clocks CLK] -max -add_delay 0.010 [get_ports
VS]

set_property CFGBVS Vcco [current_design]
```

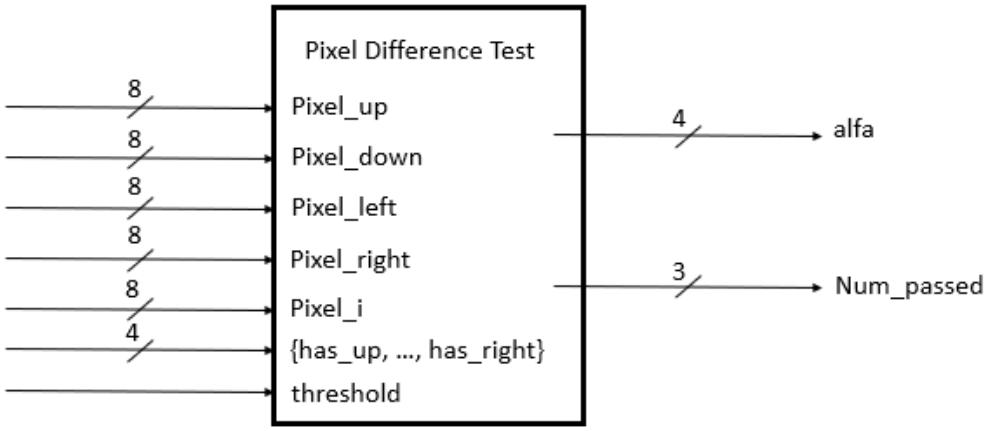
```
set_property CONFIG_VOLTAGE 3.3 [current_design]

## CLK
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property PACKAGE_PIN E3 [get_ports CLK]

## noise off switch
set_property IOSTANDARD LVCMOS33 [get_ports RST]
set_property PACKAGE_PIN P4 [get_ports RST]

##PMOD
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {spikes_line[0]}]

# JB
set_property PACKAGE_PIN U11 [get_ports {spikes_line[15]}]
set_property PACKAGE_PIN T9 [get_ports {spikes_line[14]}]
set_property PACKAGE_PIN R16 [get_ports {spikes_line[13]}]
set_property PACKAGE_PIN K16 [get_ports {spikes_line[12]}]
set_property PACKAGE_PIN V15 [get_ports {spikes_line[11]}]
set_property PACKAGE_PIN V11 [get_ports {spikes_line[10]}]
set_property PACKAGE_PIN P15 [get_ports {spikes_line[9]}]
set_property PACKAGE_PIN G14 [get_ports {spikes_line[8]}]
# JA
set_property PACKAGE_PIN E18 [get_ports {spikes_line[7]}]
set_property PACKAGE_PIN D18 [get_ports {spikes_line[6]}]
set_property PACKAGE_PIN C17 [get_ports {spikes_line[5]}]
set_property PACKAGE_PIN G13 [get_ports {spikes_line[4]}]
set_property PACKAGE_PIN E17 [get_ports {spikes_line[3]}]
set_property PACKAGE_PIN D17 [get_ports {spikes_line[2]}]
set_property PACKAGE_PIN F14 [get_ports {spikes_line[1]}]
set_property PACKAGE_PIN B13 [get_ports {spikes_line[0]}]

# LED
#set_property PACKAGE_PIN R2 [get_ports {spikes_line[14]}]
#set_property PACKAGE_PIN U1 [get_ports {spikes_line[13]}]
#set_property PACKAGE_PIN P5 [get_ports {spikes_line[12]}]
#set_property PACKAGE_PIN R1 [get_ports {spikes_line[11]}]
#set_property PACKAGE_PIN V1 [get_ports {spikes_line[10]}]
#set_property PACKAGE_PIN U3 [get_ports {spikes_line[9]}]
#set_property PACKAGE_PIN V4 [get_ports {spikes_line[8]}]
#set_property PACKAGE_PIN U6 [get_ports {spikes_line[7]}]
#set_property PACKAGE_PIN U7 [get_ports {spikes_line[6]}]
#set_property PACKAGE_PIN T4 [get_ports {spikes_line[5]}]
```

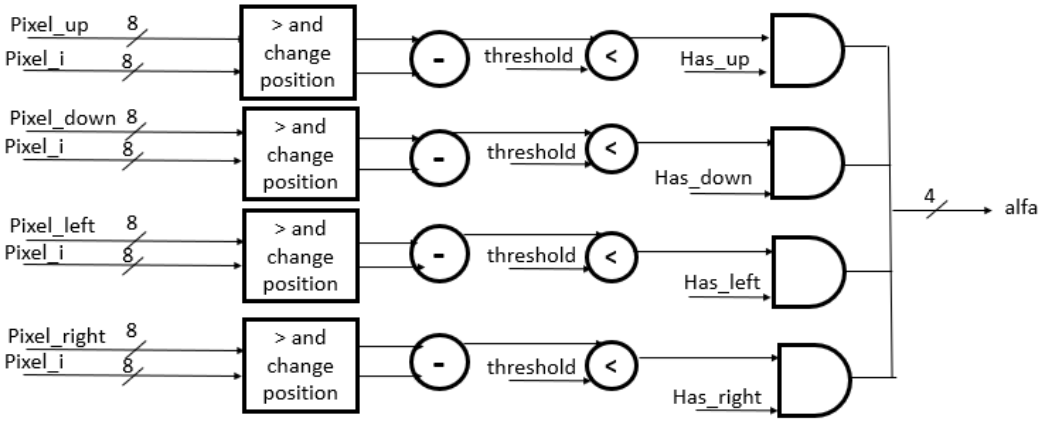
```
#set_property PACKAGE_PIN T5 [get_ports {spikes_line[4]}]
#set_property PACKAGE_PIN T6 [get_ports {spikes_line[3]}]
#set_property PACKAGE_PIN R8 [get_ports {spikes_line[2]}]
#set_property PACKAGE_PIN V9 [get_ports {spikes_line[1]}]
#set_property PACKAGE_PIN T8 [get_ports {spikes_line[0]}]

#input bottons
set_property IOSTANDARD LVCMOS33 [get_ports start]
set_property PACKAGE_PIN T16 [get_ports start]
set_property IOSTANDARD LVCMOS33 [get_ports up]
set_property PACKAGE_PIN F15 [get_ports up]
set_property IOSTANDARD LVCMOS33 [get_ports down]
set_property PACKAGE_PIN V10 [get_ports down]
set_property IOSTANDARD LVCMOS33 [get_ports ok]
set_property PACKAGE_PIN E16 [get_ports ok]
set_property IOSTANDARD LVCMOS33 [get_ports change_img]
set_property PACKAGE_PIN R10 [get_ports change_img]

# segment values
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SSEG[6]}]

set_property PACKAGE_PIN M4 [get_ports {SSEG[0]}]
set_property PACKAGE_PIN L6 [get_ports {SSEG[1]}]
set_property PACKAGE_PIN M2 [get_ports {SSEG[2]}]
set_property PACKAGE_PIN K3 [get_ports {SSEG[3]}]
set_property PACKAGE_PIN L4 [get_ports {SSEG[4]}]
set_property PACKAGE_PIN L5 [get_ports {SSEG[5]}]
set_property PACKAGE_PIN N1 [get_ports {SSEG[6]}]
set_property PACKAGE_PIN L3 [get_ports {SSEG[7]}]

#digit selection
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {CIFRA[4]}]

set_property PACKAGE_PIN N6 [get_ports {CIFRA[0]}]
set_property PACKAGE_PIN M6 [get_ports {CIFRA[1]}]
set_property PACKAGE_PIN M3 [get_ports {CIFRA[2]}]
set_property PACKAGE_PIN N5 [get_ports {CIFRA[3]}]
set_property PACKAGE_PIN N2 [get_ports {CIFRA[4]}]
set_property PACKAGE_PIN N4 [get_ports {CIFRA[5]}]
set_property PACKAGE_PIN L1 [get_ports {CIFRA[6]}]
set_property PACKAGE_PIN M1 [get_ports {CIFRA[7]}]

#switches
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[0]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel_img[14]}]
set_property PACKAGE_PIN U9 [get_ports {sel_img[0]}]
set_property PACKAGE_PIN V7 [get_ports {sel_img[5]}]
set_property PACKAGE_PIN R3 [get_ports {sel_img[13]}]
set_property PACKAGE_PIN U8 [get_ports {sel_img[1]}]
set_property PACKAGE_PIN R5 [get_ports {sel_img[4]}]
set_property PACKAGE_PIN V2 [get_ports {sel_img[9]}]
set_property PACKAGE_PIN T1 [get_ports {sel_img[12]}]
set_property PACKAGE_PIN R7 [get_ports {sel_img[2]}]
set_property PACKAGE_PIN T3 [get_ports {sel_img[11]}]
set_property PACKAGE_PIN U4 [get_ports {sel_img[8]}]
set_property PACKAGE_PIN U2 [get_ports {sel_img[10]}]
set_property PACKAGE_PIN V5 [get_ports {sel_img[7]}]
set_property PACKAGE_PIN R6 [get_ports {sel_img[3]}]
set_property PACKAGE_PIN V6 [get_ports {sel_img[6]}]
set_property PACKAGE_PIN P3 [get_ports {sel_img[14]}]

# VGA
set_property IOSTANDARD LVCMOS33 [get_ports HS]
set_property PACKAGE_PIN B11 [get_ports HS]
set_property IOSTANDARD LVCMOS33 [get_ports VS]
set_property PACKAGE_PIN B12 [get_ports VS]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DATA_OUT[5]}]
set_property PACKAGE_PIN A3 [get_ports {DATA_OUT[0]}]
set_property PACKAGE_PIN A4 [get_ports {DATA_OUT[3]}]
set_property PACKAGE_PIN C6 [get_ports {DATA_OUT[4]}]
set_property PACKAGE_PIN B4 [get_ports {DATA_OUT[1]}]
set_property PACKAGE_PIN D7 [get_ports {DATA_OUT[10]}]
set_property PACKAGE_PIN C7 [get_ports {DATA_OUT[9]}]
set_property PACKAGE_PIN D8 [get_ports {DATA_OUT[11]}]
set_property PACKAGE_PIN B7 [get_ports {DATA_OUT[8]}]
set_property PACKAGE_PIN A6 [get_ports {DATA_OUT[7]}]
set_property PACKAGE_PIN C5 [get_ports {DATA_OUT[2]}]
set_property PACKAGE_PIN B6 [get_ports {DATA_OUT[6]}]
```

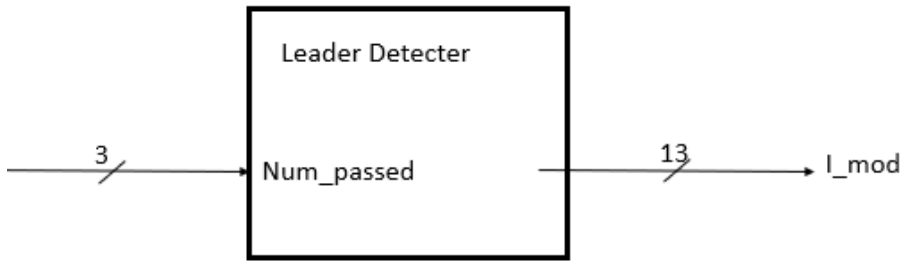
```
set_property PACKAGE_PIN A5 [get_ports {DATA_OUT[5]}]

### WNS solutions
set_property SLEW FAST [get_ports HS]
set_property SLEW FAST [get_ports VS]
set_property SLEW FAST [get_ports {DATA_OUT[7]}]
set_property SLEW FAST [get_ports {DATA_OUT[2]}]
set_property SLEW FAST [get_ports {DATA_OUT[5]}]
set_property SLEW FAST [get_ports {DATA_OUT[6]}]
set_property SLEW FAST [get_ports {DATA_OUT[8]}]
set_property SLEW FAST [get_ports {DATA_OUT[11]}]
set_property SLEW FAST [get_ports {DATA_OUT[9]}]
set_property SLEW FAST [get_ports {DATA_OUT[4]}]
set_property SLEW FAST [get_ports {DATA_OUT[1]}]
set_property SLEW FAST [get_ports {DATA_OUT[10]}]
set_property SLEW FAST [get_ports {DATA_OUT[3]}]
set_property SLEW FAST [get_ports {DATA_OUT[0]}]

set_property SLEW FAST [get_ports {CIFRA[1]}]
set_property SLEW FAST [get_ports {CIFRA[6]}]
set_property SLEW FAST [get_ports {CIFRA[3]}]
set_property SLEW FAST [get_ports {CIFRA[0]}]
set_property SLEW FAST [get_ports {CIFRA[5]}]
set_property SLEW FAST [get_ports {CIFRA[2]}]
set_property SLEW FAST [get_ports {CIFRA[7]}]
set_property SLEW FAST [get_ports {CIFRA[4]}]
set_property SLEW FAST [get_ports {SSEG[1]}]
set_property SLEW FAST [get_ports {SSEG[4]}]
set_property SLEW FAST [get_ports {SSEG[0]}]
set_property SLEW FAST [get_ports {SSEG[6]}]
set_property SLEW FAST [get_ports {SSEG[3]}]
set_property SLEW FAST [get_ports {SSEG[2]}]
set_property SLEW FAST [get_ports {SSEG[5]}]
set_property SLEW FAST [get_ports {SSEG[7]}]

set_property SLEW FAST [get_ports {spikes_line[15]}]
set_property SLEW FAST [get_ports {spikes_line[14]}]
set_property SLEW FAST [get_ports {spikes_line[13]}]
set_property SLEW FAST [get_ports {spikes_line[12]}]
set_property SLEW FAST [get_ports {spikes_line[11]}]
set_property SLEW FAST [get_ports {spikes_line[10]}]
set_property SLEW FAST [get_ports {spikes_line[9]}]
set_property SLEW FAST [get_ports {spikes_line[8]}]
set_property SLEW FAST [get_ports {spikes_line[7]}]
set_property SLEW FAST [get_ports {spikes_line[6]}]
set_property SLEW FAST [get_ports {spikes_line[5]}]
set_property SLEW FAST [get_ports {spikes_line[4]}]
set_property SLEW FAST [get_ports {spikes_line[3]}]
set_property SLEW FAST [get_ports {spikes_line[2]}]
set_property SLEW FAST [get_ports {spikes_line[1]}]
set_property SLEW FAST [get_ports {spikes_line[0]}]
```

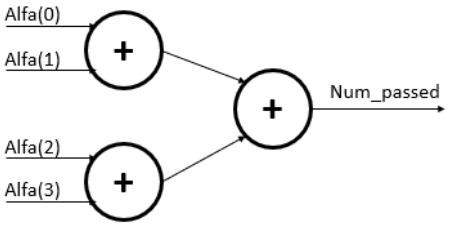
A3. Esquema de los bloques que componen al sistema

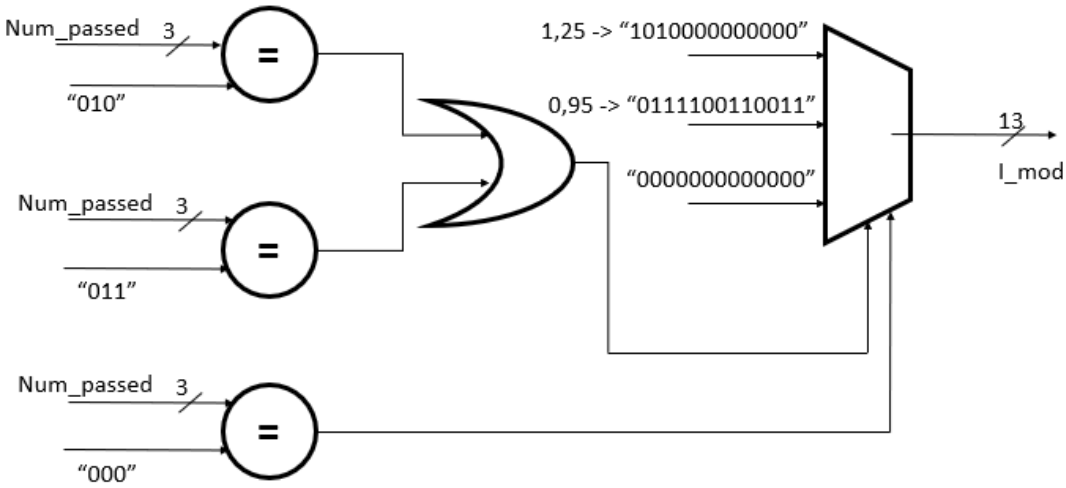




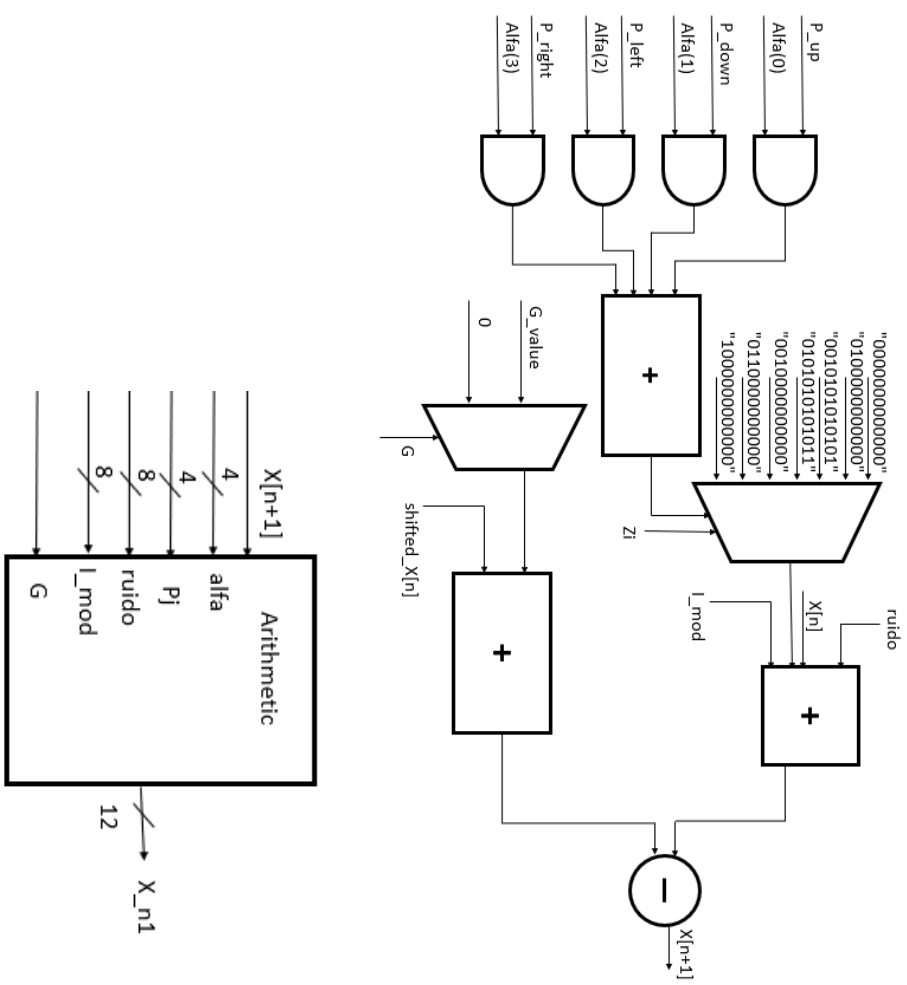
Sistema electrónico para la realización flexible de redes neuronales		
Esquema nº: **1**	**Título de esquema:** **Esquema de la unidad pixel difference test**	
	Nombre del autor	**Fecha**
Creado	**Xinqiu Ye**	**01-05-2018**
Revisado	**Xinqiu Ye**	**25-05-2018**



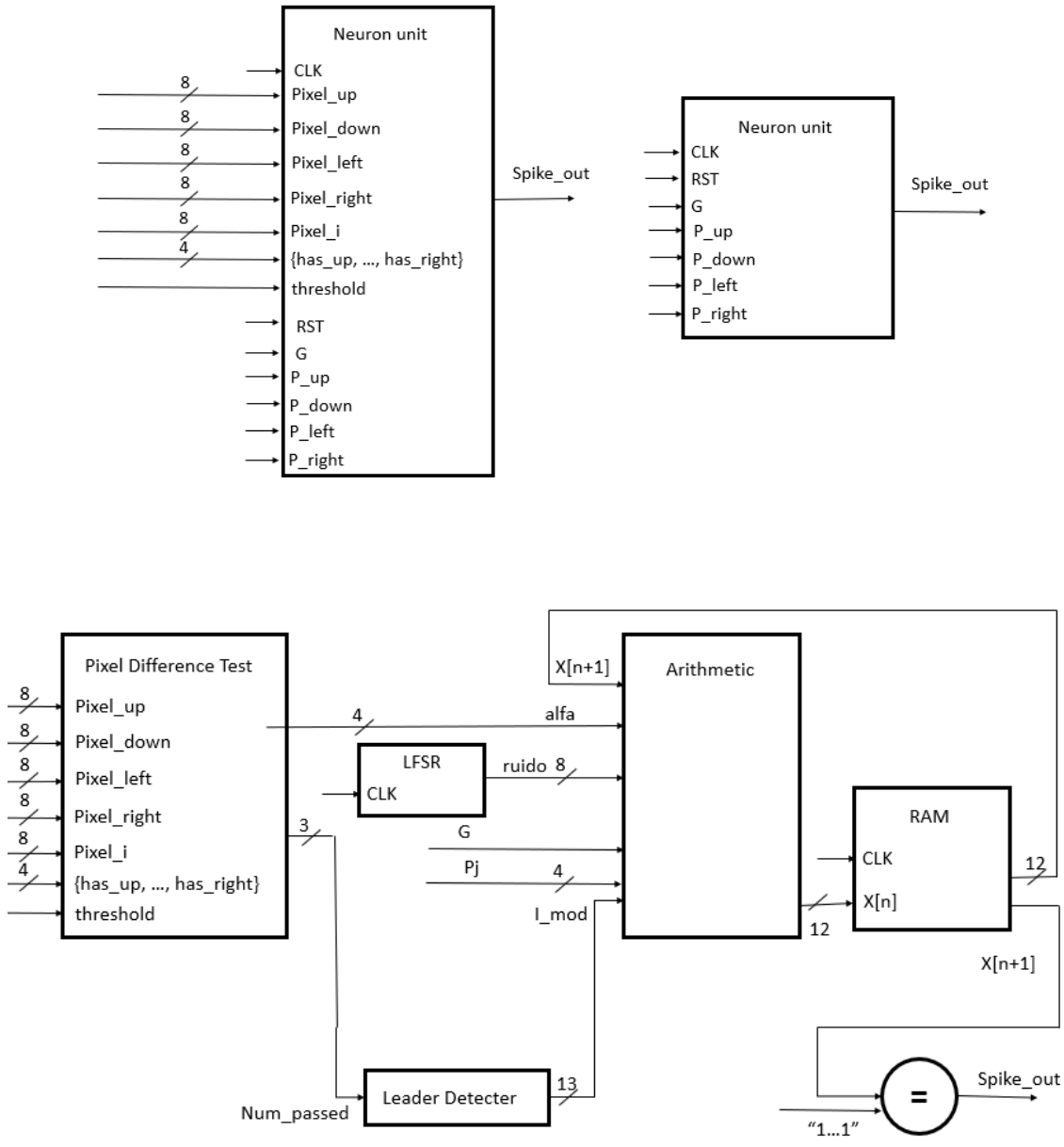




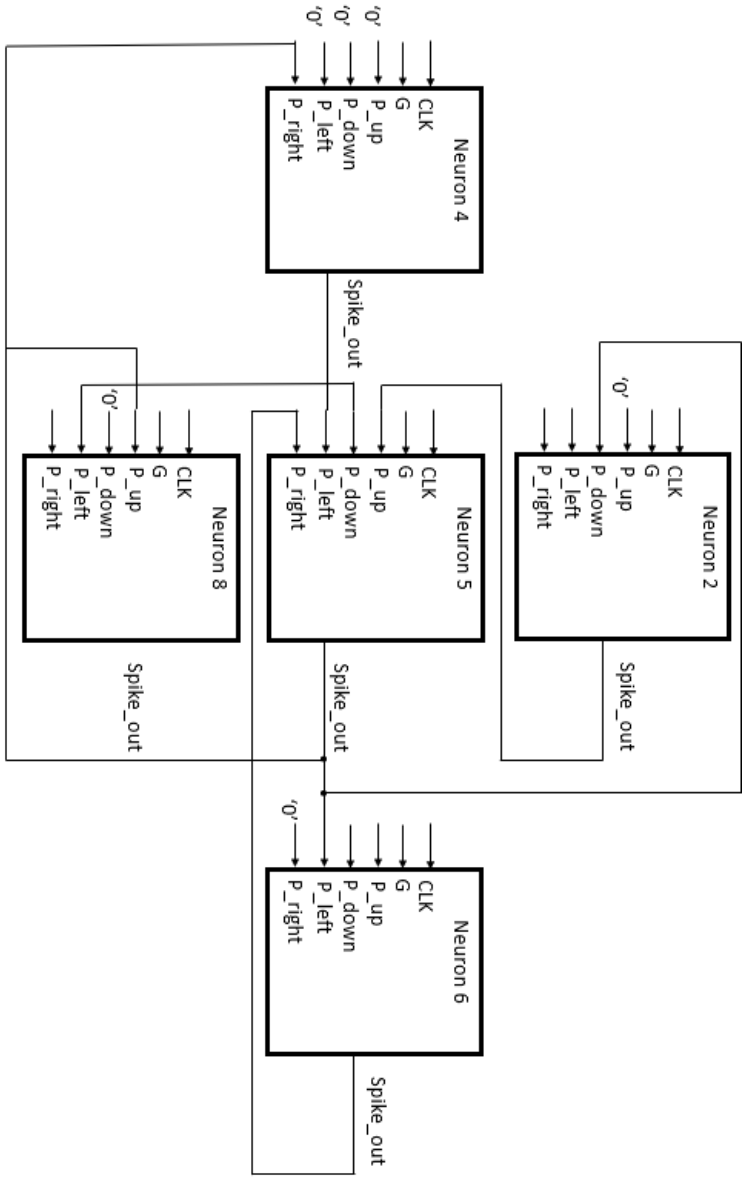
Sistema electrónico para la realización flexible de redes neuronales		
Esquema nº: **2**	**Título de esquema:** **Esquema de la unidad *leader detecter***	
	Nombre del autor	**Fecha**
Creado	**Xinqiu Ye**	**01-05-2018**
Revisado	**Xinqiu Ye**	**25-05-2018**



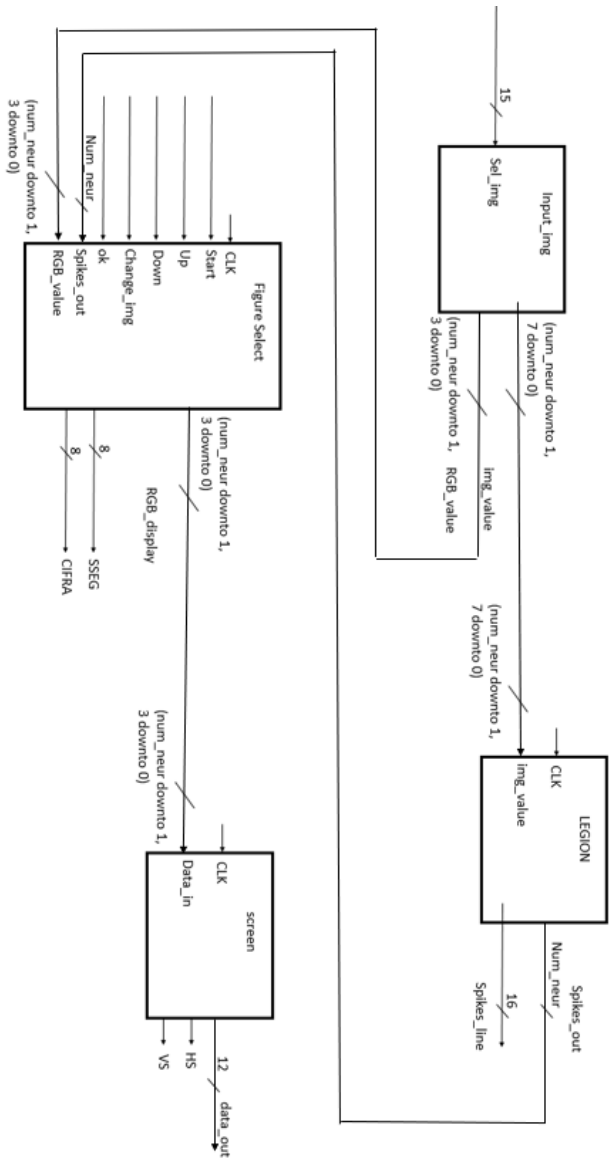
Sistema electrónico para la realización flexible de redes neuronales		
Esquema nº: **3**	**Título de esquema:** **Esquema de la unidad aritmética**	
	Nombre del autor	**Fecha**
Creado	**Xinqiu Ye**	**01-05-2018**
Revisado	**Xinqiu Ye**	**25-05-2018**



Sistema electrónico para la realización flexible de redes neuronales		
Esquema nº: **4**	**Título de esquema:** **Esquema de la arquitectura general de una neurona**	
	Nombre del autor	**Fecha**
Creado	**Xinqiu Ye**	**01-05-2018**
Revisado	**Xinqiu Ye**	**25-05-2018**



Sistema electrónico para la realización flexible de redes neuronales		
Esquema nº: **5**	**Título de esquema:** **Interconexión de 4 osciladores en la red**	
	Nombre del autor	**Fecha**
Creado	**Xinqiu Ye**	**01-05-2018**
Revisado	**Xinqiu Ye**	**25-05-2018**



Sistema electrónico para la realización flexible de redes neuronales		
Esquema nº: **6**	**Título de esquema:** **Arquitectura general del sistema electrónico diseñado**	
	Nombre del autor	**Fecha**
Creado	**Xinqiu Ye**	**01-05-2018**
Revisado	**Xinqiu Ye**	**25-05-2018**